A dissertation

submitted in fulfillment of the requirements for the degree of doctor of philosophy in Computer Science and Engineering

Development of a Novel Algorithm and an Evaluation Method for Lossless Data Compression

by

Md Atiqur Rahman

Graduate Department of Computer and Information Systems

The University of Aizu

2022

The thesis titled

Development of a Novel Algorithm and an Evaluation Method for Lossless Data Compression

by

Md Atiqur Rahman

is reviewed and approved by:

Main referee

Senior Associate Professor, The University of Aizu, Japan

HAMADA Mohamed *Signature:* 浜田

Professor, The University of Aizu, Japan

SHIN Jungpil *Signature:* 慎

Senior Associate Professor, The University of Aizu, Japan

YOSHIOKA Rentaro *Signature:* 吉岡

Senior Associate Professor, The University of Aizu, Japan

WATANOBE Yutaka *Signature:* 渡部

THE UNIVERSITY OF AIZU

2022

Abstract

Lossless data compression is a topic of great research interest in the field of computing, especially for applications that depend on low bandwidth connections and limited storage. There are various types of images, and many algorithms are used to compress these losslessly. The performance of a lossless data compression algorithm depends on the compression ratio (CR), encoding time (ET), and decoding time (DT). First, a transformation or prediction technique is applied to an image, and then an entropy coding method for image compression.

In this thesis, we study and analyze the entropy coding techniques and recommend an entropy coding technique as the best. Also, we show which parts of the algorithms need to be improved.

Most of the research works compare the state-of-the-art techniques based on compression ratio, encoding time, or decoding time to evaluate the effectiveness of an algorithm. While a higher compression ratio is more important for some applications, others may require faster encoding or decoding, or both. Alternatively, each of the three parameters can be equally significant. Therefore, choosing an optimal algorithm from many algorithms based on an application's requirements is a significant challenge. This thesis proposed a model (PCBMS) that predicts an algorithm as the best by analyzing the data from each perspective. However, a better prediction depends on making a good balance between compression ratio, encoding, and decoding times. Therefore, we proposed an alternative approach to PCBMS that can balance the parameters better than PCBMS and gives more accurate predictions.

In terms of text compression, many techniques use Burrows-Wheeler transform and run-length coding as part of compression. We present a different approach for text compression that uses keys instead of the run-length technique during the coding of characters' length.

We also discuss the state-of-the-art lossless data compression techniques in detail and finally present some problems as future research directions.

Acknowledgment

First and foremost, I am incredibly grateful to my supervisor, Prof. Mohamed Hamada, for his invaluable advice, continuous support, and patience during my Ph.D. study. His immense knowledge and great experience have encouraged me in my academic research and daily life. I am also grateful to him for his valuable feedback and suggestions to achieve my research goals.

I am highly indebted to pass my heartfelt thanks to JASSO (Japan Student Services Organization) for the financial support that has made my study and life in Japan a wonderful time.

My gratitude goes to my supervisory committee member Prof. Jungpil Shin, Prof. Rentaro Yoshioka, and Prof. Yutaka Watanobe for their valuable comments and feedback on this dissertation. I want to thank all my friends, lab fellows, and companions in this journey.

Finally, I would like to express my gratitude to my parents, my wife, and my children. Without their tremendous understanding and encouragement in the past few years, completing my study would have been impossible.

Dedicated to my family.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
BCH	BoseChaudhuri-Hocquenghem
LZW	Lempel–Ziv–Welch
ANN	Artificial Neural Network
DCT	Discrete Cosine Transform
iDCT	Integer Discrete Cosine Transform
DTT	Discrete Tchebichef Transform
iDTT	Integer Discrete Tchebichef Transform
ROI	Region of Interest
IWLCA	Improved Wavelet Lossless Compression Algorithm
DWT	Discrete Wavelet Transform
DFrFT	Discrete Fractional Fourier Transform
CR	Compression Ratio
ET	Encoding Time
DT	Decoding Time
RGB	Red,Green, and Blue
SCIE	Science Citation Index Expanded
IEEE	Institute of Electrical and Electronics Engineers
BWT	Burrows–Wheeler Transform
HSV	Hhue, Saturation, and Value
CMYK	Cyan, Magenta, Yellow, and Key (black)
HDTV	High-Definition Television
SHV	Super Hi-Vision
GB	Gigabyte
TB	Terabyte
HDDs	Hard Disk Drives
Gbps	Gigabits per second
AIC	Average Information Content
DFT	Discrete Fourier Transform
MSE	Mean Squared Error
SNR	Signal-to-noise ratio

Abbreviation	**Definition**
PSNR	Peak signal-to-noise ratio
MSD	Mean Squared Deviation
DC	Transform Coefficient
AC	Joint Photographic Experts Group
JPEG	Differential Pulse Code Modulation
DPCM	Median Edge Detector
MED	Joint Photographic Experts Group-Lossless Standard
JPEG-LS	Low Complexity Lossless Compression for Images
LOCO-I	two-sided geometric distribution
PNG	Portable Network Graphics
GIF	Graphics Interchange Format
CALIC	Context-Based, Adaptive, Lossless Image Code
GAP	Gradient-adjusted predictor
FLIF	Free Lossless Image Forma
EBCOT	Embedded block coding with optimal truncation
JPEG XR	JPEG extended range
PCT	Photo core transform
QF	Quantisation factor
POT	Photo overlap transform
AVIF	AV1 Image File Format
SVD	Singular Value Decomposition
TIFF	Tag Image File Format
PDF	Portable Document Format
MRI	Magnetic resonance imaging
RLE	Run-length encoding
OVP	Overall performance
GTP	Grand total performance
LZMA	Lempel–Ziv–Markov chain algorithm
PCBMS	Parameter combination-based method selection
bpsp	Bits per sub pixel
CBP	Context-based bit-plane codec
HEVC	High Efficiency Video Coding

Chapter 1

Introduction

Digital data compression is an interesting feature of today's advanced technology for storage, transmission, and representation of autonomous machine perception. This chapter has several objectives: (1) the background of the digital data compression; (2) our objectives and contributions.

1.1 Background

The demand for digital information has increased dramatically over the past few decades with the development of multimedia technology. Advances in technology have largely increased the use of digital imagery. Still images are widely used in applications like digital radiography, scientific imaging, zip file compression, museums/art galleries, medical, satellite images, etc. Digital images consist of a huge amount of data. Reducing image size is becoming increasingly important for both storing and transmitting digital images as they get more applications. Image compression is a mapping from a higher-dimensional space to a lower-dimensional space. Image compression plays a vital role in several multimedia applications, such as image transmission and storage. The primary goal of image compression is to represent an image with a minimum number of bits of acceptable image quality. All image compression techniques try to eliminate statistical redundancy and use conceptual irrelevance while minimizing data volume as much as possible.

With the advancement of the Internet, teleconferencing, multimedia, and high-definition television technologies, the amount of information that computers handle has increased significantly over the past few decades. Thus, storage and transmission of the digital image component of multimedia systems is a significant problem. The amount of information needed to present images at a satisfactory level of quality is extremely high. High-quality image data requires a large amount of storage space and transmission bandwidth, which sometimes current technology is technically and economically incapable of managing. One possible solution to this problem is to compress data to reduce storage space and transmission time.

The amount of information transferred over the internet doubles every year, and a considerable portion of that information contains images. Reducing the bandwidth requires of any device will result in significant cost reductions and make the device more affordable. Image compression offers a way to represent an image more compactly so that images can be stored compactly and transmitted quickly. The images are highly coherent meaning there is redundant information here. Compression is gained through redundancy and irrelevancy reduction. Redundancy means duplication and irrelevancy means the part of the image information that the human visual system will not notice. Every data compression technique compresses data in two steps: transformation/prediction and then entropy coding.

Alarabeyyat et al. in [5] proposed a lossless image compression technique using Bose-Chaudhuri-Hocquenghem (BCH) and Lempel–Ziv–Welch (LZW) to improve the compression ratio. They showed that the proposed algorithm provided a better compression ratio of 11.65% and 28.66% compared to LZW and Huffman coding. In [6], a lossless image compression method using an artificial neural network (ANN) and Huffman coding was proposed. The compression procedure was tested by three datasets, namely CLEF med 2009, COREL1 k, and standard benchmarking images. The proposed strategy clearly showed better compression ratios compared to the other techniques mentioned. OwenZhao et al. proposed a scheme called super-spatial structure prediction for lossless image compression in [7]. They were motivated by motion prediction in video coding. They showed that the proposed method outperforms the state-of-the-art lossless image compression techniques in terms of compression ratio. Discrete Cosine Transform (DCT) is a standard transform technique used in lossy image compression, and integer Discrete Cosine Transform (iDCT) is used for lossless compression. On the other hand, Discrete Tchebichef Transform (DTT) is an orthogonal transform used for lossy image compression. In [8], Xiao et al. introduced a lossless image compression technique called integer DTT (iDTT). The article showed that the iDTT method provides higher compression ratios than iDCT. In [9], Zuo et al. proposed a medical image compression technique based on the region of interest (ROI). In this paper, an image is divided into two parts: ROI and non-ROI regions. A lossless compression technique is applied on the ROI regions, and a wavelet-based lossy compression algorithm is applied on the non-ROI regions. The proposed approach is compared to JPEG 2000, JPEG-LS, and CALIC. It showed better results in terms of compression ratio than the methods mentioned. The wavelet transform divides an image into different but interrelated multiresolution and multi-level sub-bands that help to reduce an image more. Jio Li introduced Hilbert and singular value truncating to wavelet and proposed an Improved Wavelet Lossless Compression Algorithm (IWLCA) in [10]. It was showed in the paper that the proposed technique provides better compression than JPEG-LS and JPEG 2000. Naveen Kumar et al. proposed a lossless image compression algorithm using a combination of two-dimensional Discrete wavelet transform (DWT) and one-dimensional discrete fractional Fourier transform (DFrFT) in [11]. In the first stage of the method, an image is divided into low and high frequency sub-bands by applying the Daubechies wavelet filter. Secondly, level 1 quantization is applied for both low and high frequency sub-bands. The fractional Fourier transform is used to compress the low-frequency sub-bands, and high-frequency sub-bands are reduced by dropping zeroes and storing only nonzero blocks and its position. The compressed wavelet coefficients are further compressed using of level 2 quantization. Finally, arithmetic encoder followed by run-length coding is applied for encoding. The article showed that the proposed algorithm has significantly improved the compression ratio compared to DFrCT and DFrST. A novel prediction technique was presented in [12] that treats image data as an interleaved sequence produced by various sources. Finally, a lossless color image compression technique was proposed using the prediction technique combining with template-matching prediction and a blending approach. The proposed method also showed a better compression performance for different types of color images.

There are many entropy coding techniques. Different compression algorithms use different types of entropy coding techniques. In our studies, we have seen that a good entropy coding technique can give a good compression performance. Various compression algorithms show different performances in different types of images. From the above study, we have seen that although almost all the research works have generally evaluated each algorithm based on the compression ratio [13–26], the compression speed of the algorithm is also an important issue. While compression ratios are essential for some applications, there are many algorithms where encoding or decoding speed is necessary. Alternatively, in some applications, all three or any two are very important. So, it is essential for users to know which compression algorithm will work best on which type of image, and that is very challenging. Our study found that text compression algorithms that use run-length coding for compression can not show a good compression performance because the use of run-length coding increases the number of unique symbols and the number of characters during the length coding.

1.2 Objectives

This thesis deals with data compression that addresses four problems. The first, second, and third problems are related to choosing the best entropy coding technique, calculating the impact of the state-of-the-art lossless data compression techniques, and alternative key-based text compression technique instead of using run-length based method.

1. Choosing an entropy coding technique
    (a) **Problem 1:** Which is the best entropy coding technique?
    (b) **Problem 2:** What are the limitations of the entropy coding techniques?
    (c) **Problem 3:** Which part of the algorithms needs to be improved?
2. Selecting an Optimal Lossless Image Compression Technique
    (a) **Problem 4:** The performance of a lossless data compression algorithm depends on all parameters (bpp, encoding and decoding time) and does not singly depend on any of them. Some techniques require more time for encoding than for decoding. Some provide lower bpp than others. Therefore, the most important questions are: 1. which technique is better when any two of the parameters (compression ratio, encoding, and decoding times) are equally significant for an application, or when all parameters are equally valuable?; 2. is there any such technique that works well for all types of data? Therefore, which technique is better for eight-bit RGB, eight-bit grayscale, 16-bit RGB, 16-bit grayscale or binary or indexed images, among others?
3. Impact of the state-of-the-art lossless image compression techniques
    (a) **Problem 5:** How good is each algorithm in terms of the CR for 8-bit and 16-bit greyscale and RGB images?
    (b) **Problem 6:** How good is each algorithm in terms of the ET for 8-bit and 16-bit greyscale and RGB images?
    (c) **Problem 7:** How good is each algorithm in terms of the DT for 8-bit and 16-bit greyscale and RGB images?
    (d) **Problem 8:** How good is each algorithm in terms of the CR and ET for 8-bit and 16-bit greyscale and RGB images?
    (e) **Problem 9:** How good is each algorithm in terms of the CR and DT for 8-bit and 16-bit greyscale and RGB images?
    (f) **Problem 10:** How good is each algorithm in terms of the ET and DT for 8-bit and 16-bit greyscale and RGB images?
    (g) **Problem 11:** How good is each algorithm when all parameters are equally important for 8-bit and 16-bit greyscale and RGB images?
    (h) **Problem 12:** Which algorithms should be used for each kind of image?
4. Develop a lossless text compression method
    (a) **Problem 13:** Applying run-length coding for text compression increases the number of unique symbols, which increases the use of storage space.

1.3 Contributions

Contributions to this thesis are in the form of three published journal papers and one has been submitted or in the process of being submitted that cover the objectives mentioned above. The list of papers included in this dissertation is given in section 1.3.1. The contributions are shown in three sections, where each section is directed by one of the proposed research problems. Every section will later be extended to a chapter.

1.3.1 List of Publications

Journals

1. **Md. Atiqur Rahman**, and Mohamed Hamada, 2019. Lossless image compression techniques: A state-of-the-art survey. Symmetry, 11(10), p.1274. **(SCIE)**

2. **Md. Atiqur Rahman**, and Mohamed Hamada, 2021. PCBMS: A Model to Select an Optimal Lossless Image Compression Technique. IEEE Access, DOI: 10.1109/ACCESS.2021.3137345. **(SCIE)**

3. **Md. Atiqur Rahman**, Mohamed Hamada, and Jungpil Shin 2021. The Impact of State-of-the-Art Techniques for Lossless Still Image Compression. Electronics, 10(3), p.360. **(SCIE)**

4. **Md. Atiqur Rahman**, and Mohamed Hamada, 2020. Burrows–Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding. Symmetry, 12(10), p.1654. **(SCIE)**

5. **Md. Atiqur Rahman**, and Mohamed Hamada, 2021. A prediction-based lossless image compression procedure using dimension reduction and Huffman coding, Multimedia Tools and Applications, Springer. (Accept with minor revision) **(SCIE)**

Conferences

1. **Md. Atiqur Rahman**, Mohamed Hamada, and Md Asfaqur Rahman, 2021, December. Text compression based on an alternative approach of run-length coding using Burrows-Wheeler transform and arithmetic coding, In 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC).

2. **Md. Atiqur Rahman**, Mohamed Hamada, and Md Asfaqur Rahman, 2022, A comparative analysis of the state-of-the-art lossless image compression techniques. The 4th ETLTC International Conference on Information and Communications Technology (ETLTC 2022). (Submitted)

3. **Md. Atiqur Rahman**, and Mohamed Hamada, 2019, October. A semi-lossless image compression procedure using a lossless mode of JPEG. In 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (pp. 143-148). IEEE.

4. **Md. Atiqur Rahman**, and Mohamed Hamada, 2021. Lossless text compression using GPT-2 language model and Huffman coding. In Proceedings of The 2021 3rd ETLTC - ACM Chapter International Conference on Information and Communications Technology.

5. **Md. Atiqur Rahman**,Md Faizul Ibne Amin,and Mohamed Hamada, 2020, August. Edge Detection Technique by Histogram Processing with Canny Edge Detector. In 202020 3rd

IEEE International Conference on Knowledge Innovation and Invention (ICKII) (pp. 128-131). IEEE.

6. **Md. Atiqur Rahman**, Most. Jannatul Ferdous, Md. Mamun Hossain, Md Rashedul Islam, and Mohamed Hamada, 2019, May. A lossless speech signal compression technique. In 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT) (pp. 1-7). IEEE.

1.3.2 Choosing the best entropy coding technique (Problem 1-3)

Publication 1: Lossless Image Compression Techniques: A State-of-the-Art Survey [27].

Contributions: Some of the entropy coding techniques provide better compression than the other, while the other method takes less encoding and decoding time. Some algorithms are very sensitive to noise. Thus, choosing an entropy coding strategy for lossless data compression is a big challenge. We have analyzed entropy coding techniques in detail and propose an entropy coding strategy as the best among them. We finally show the limitations of the algorithms and show which part of the algorithms needs to be improved.

1.3.3 Selecting an Optimal Lossless Image Compression Technique (Problem 4)

Publication 2: PCBMS: A Model to Select an Optimal Lossless Image Compression Technique. [28]

Contributions: A mathematical model to select an optimal lossless image compression technique is proposed in this chapter. This chapter shows that each algorithm was evaluated based on a specific parameter in each research work. However, the performance of a lossless image compression algorithm depends on all parameters (bpp, encoding and decoding time) and does not singly depend on any of them. Therefore, the proposed method predicts a better lossless image compression algorithm for any combination of parameters and provides the actual impact of each algorithm.

1.3.4 Impact of the state-of-the-art techniques (Problem 5-12)

Publication 3: The Impact of State-of-the-Art Techniques for Lossless Still Image Compression [29].

Contributions: The PCBMS model gives a better prediction to select a better lossless image compression method. However, a better prediction depends on making a good balance between compression ratio, encoding, and decoding times. Therefore, we proposed an alternative approach to PCBMS. The proposed method can balance the parameters better than PCBMS and give more accurate predictions.

1.3.5 Develop a lossless text compression method (Problem 13)

Publication 4: Burrows–Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding [30].

Publication 4: Lossless text compression using GPT-2 language model and Huffman coding [31].

Contributions: Many advanced text compression techniques use Burrows-Wheeler transform, run-length coding, pattern finding and then an entropy coding technique, respectively, for compression. Run-length coding increases the number of unique symbols during the coding of characters' length. As a result, the compression ratio is reduced. To solve the problem, we propose a key-based coding technique in place of run-length coding that increases the compression ratio.

1.4 Thesis organization

This thesis is based on eight manuscripts, and they are grouped into six Chapters.

1. Chapter 2 gives a basic idea about data compression, how to compress, and the metrics used to evaluate a data compression technique.

2. Chapter 3 explains the transformation techniques, such as Discrete Wavelet Transform (DWT), Haar Wavelet, Daubechies-4 Wavelet, and Burrows-Wheeler Transform (BWT).

3. Chapter 4 describes the state-of-the-art data compression techniques, such as run-length coding, Shannon-Fano Coding, Huffman coding, Lempel–Ziv–Welch (LZW) Coding, Arithmetic Coding, Lossless JPEG, JPEG-LS, PNG, CALIC, WebP, FLIF, JPEG 2000, JPEG XR, AVIF, and LZ77.

4. Chapter 5 contains a journal paper. This chapter analyzes the entropy coding techniques, and proposes the best entropy coding technique. It also shows which part of the methods needs to be improved.

5. Chapter 6 contains a journal paper. In this chapter, an evaluation technique (PCBMS) is proposed that predicts an optimal lossless image compression method as the best.

6. Chapter 7 consists of a journal paper. In this chapter, we proposed an alternative approach to PCBMS that can give a better prediction than PCBMS for each type of image, based on users' particular needs.

7. Chapter 8, which contains a journal paper where a key-based lossless text compression method is proposed.

8. Chapter 9 concludes the thesis and also some future works. A summary of the thesis is also given.

Chapter 2

Data Compression

2.1 Introduction

A visual representation of an object is called an image, and a digital image can be defined as a two-dimensional matrix of discrete values. When the colour at each position in an image is represented as a single tone, this is referred to as a continuous tone image. The quantised values of a continuous tone image at discrete locations are called the grey levels or the intensity [32], and the pixel brightness of a digital image is indicated by its corresponding grey level. The steps used to transform a continuous tone image to its digital form are shown in Figure 2.1.

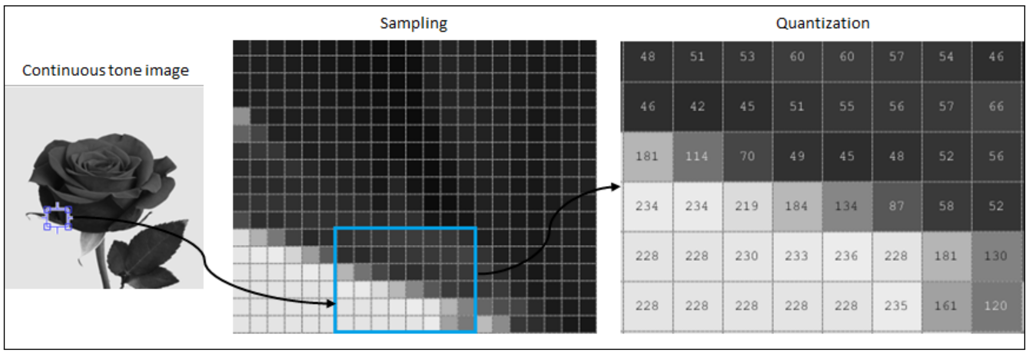
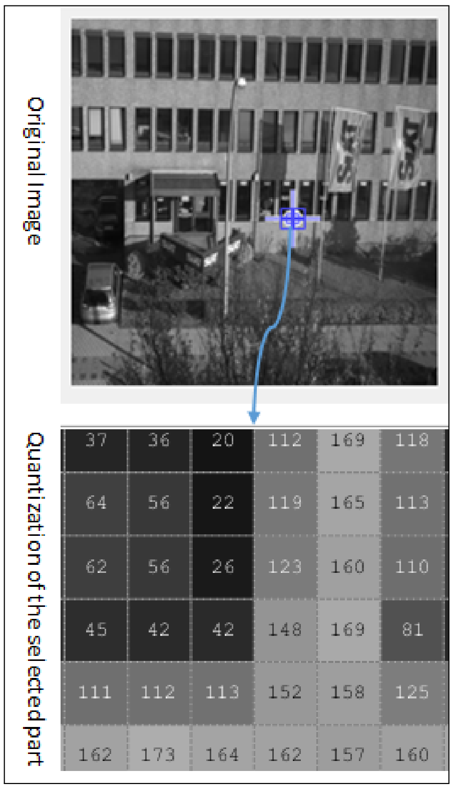


Figure 2.1: Steps used to convert a continuous tone image to a digital one

The bit depth indicates the number of bits used to represent a pixel, where a higher bit depth represents more colours, thus increasing the file size of an image [33]. A greyscale image is a matrix of AxB pixels, and 8-bit and 16-bit greyscale images contain $2^8 = 256$ and $2^{16} = 65536$ different colours, respectively, where the ranges of colour values are from 0–255 and 0–65535. Examples of 8-bit and 16-bit greyscale images are shown in Figures 2.2 and 2.3, respectively.

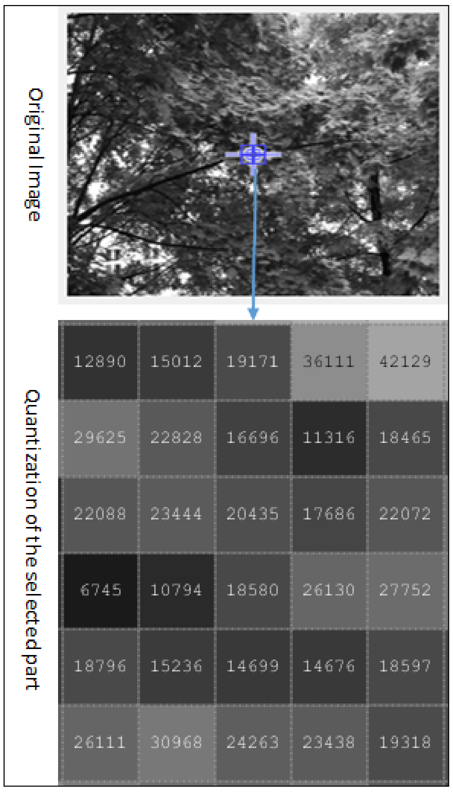
A particular way of representing colours is called the colour space, and a colour image is a linear combination of these colours. There are many colour spaces, but the most popular are RGB, HSV and CMYK. RGB contains the three primary colours of red, green and blue, and is used by computer monitors. HSV (hue, saturation, value) and CMYK (cyan, magenta, yellow, and key (black)) are often used by artists and in the printing industry, respectively [32]. A colour image carries three colours per pixel; for example, since an RGB image uses red, green and blue, each pixel of an 8-bit RGB image has a precision of 24 bits, and the image can represent $2^{24} = 16,777,216$ different shades. For a 16-bit RGB image, each pixel has a precision of 48 bits, allowing for $2^{48} = 281,474,976,710,656$ different shades. Typical examples of 8-bit and 16-bit RGB images are shown in Figures 2.4 and 2.5, respectively. The ranges of colour values for 8-bit and 16-bit images are 0–255 and 0–65535, respectively.

For an uncompressed image (X), the memory required to store the image is calculated using



37	36	20	112	169	118
64	56	22	119	165	113
62	56	26	123	160	110
45	42	42	148	169	81
111	112	113	152	158	125
162	173	164	162	157	160

Figure 2.2: An 8-bit greyscale image



12890	15012	19171	36111	42129
29625	22828	16696	11316	18465
22088	23444	20435	17686	22072
6745	10794	18580	26130	27752
18796	15236	14699	14676	18597
26111	30968	24263	23438	19318

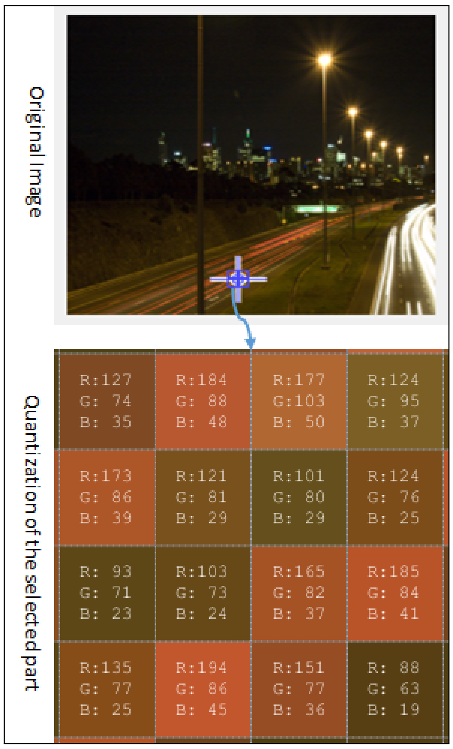
Figure 2.3: A 16-bit greyscale image

Equation 2.1, where the dimensions of the image are AxB and the bit depth is N.

$$Storage = AxBxNx2^{-13}KB \tag{2.1}$$

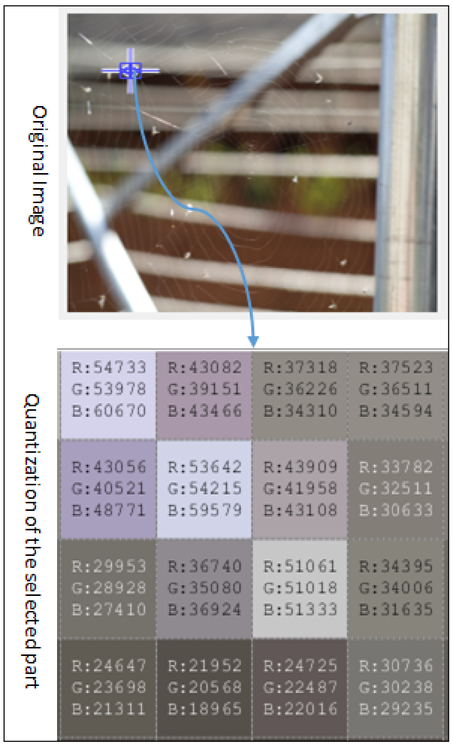
2.2 How data is compressed

Data compression is a significant issue and a subject of intense research in the field of multimedia processing. We give a real example below to allow for a better understanding of the importance of data compression. Nowadays, digital cinema and high-definition television (HDTV) use a 4K system, with approximately 4096x2160 pixels per frame [34]. However, the newly developed Super Hi-Vision (SHV) format uses 7680x4320 pixels per frame, with a frame rate of 60 frames per second [35]. Suppose we have a three-hour colour video file based on SHV video technology, where each pixel has a precision of 48 bits. The size of the video file will then be (7680x4320x48x60x3x60x60) bits, or approximately 120,135.498 GB. According to the report in [36], 500 GB to 1 TB is appropriate for storing movies for non-professional users. Seagate, an American data storage company, has published quarterly statistics since 2015 on the average capacity of Seagate hard disk drives (HDDs) worldwide. In the third quarter of 2020, this capacity was 4.1 TB [37]. Can we imagine what would have happened? We couldn't even store a three-hour color SHV video file on our local computer. Compression is another important issue for data transmission over the internet. Although there are many forms of media that can be used for transmission, fibre optic cables have the highest transmission speed [38], and can transfer up to 10 Gbps [39]. If this video file is transferred at the highest speed available over fibre optic media without compression, approximately 26.697 hours would be required, without considering latency. Latency is the amount of time required to transfer data from the original source to the destination [40]. In view of the above problems, current technology is entirely inadequate, and the only effective solution is data compression.



R:127 G: 74 B: 35	R:184 G: 88 B: 48	R:177 G:103 B: 50	R:124 G: 95 B: 37
R:173 G: 86 B: 39	R:121 G: 81 B: 29	R:101 G: 80 B: 29	R:124 G: 76 B: 25
R: 93 G: 71 B: 23	R:103 G: 73 B: 24	R:165 G: 82 B: 37	R:185 G: 84 B: 41
R:135 G: 77 B: 25	R:194 G: 86 B: 45	R:151 G: 77 B: 36	R: 88 G: 63 B: 19

Figure 2.4: An 8-bit RGB image



R:54733 G:53978 B:60670	R:43082 G:39151 B:43466	R:37318 G:36226 B:34310	R:37523 G:36511 B:34594
R:43056 G:40521 B:48771	R:53642 G:54215 B:59579	R:43909 G:41958 B:43108	R:33782 G:32511 B:30633
R:29953 G:28928 B:27410	R:36740 G:35080 B:36924	R:51061 G:51018 B:51333	R:34395 G:34006 B:31635
R:24647 G:23698 B:21311	R:21952 G:20568 B:18965	R:24725 G:22487 B:22016	R:30736 G:30238 B:29235

Figure 2.5: A 16-bit RGB image

An image is a combination of information and redundant data, as shown in Figure 2.6. One of the most important issues in image compression is how much information an image contains. If an image contains a number of unique symbols SL, and P(k) is the probability of the kth symbol, the average information content (AIC) that an image may contain, also known as entropy, is calculated using Equation 2.2. Image compression is achieved through a reduction in redundant data.

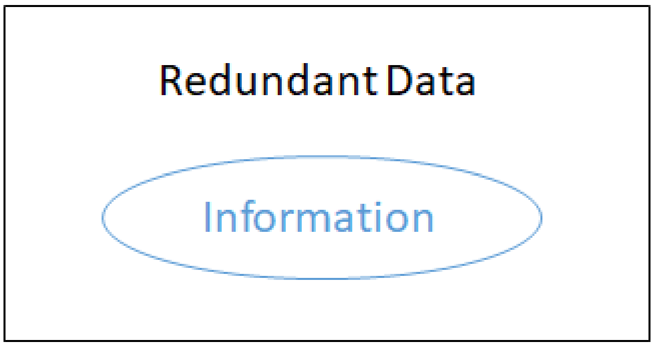


Figure 2.6: Redundant data in an image

$$AIC = -\sum_{k=1}^{SL} log(P(k)P(k)) \tag{2.2}$$

Suppose two datasets A and B point to the same image or information. Equation 2.3 can then be used to define the relative data redundancy (R_{dr}) of set A, where the CR is calculated using Equation 2.4.

$$R_{dr} = 1 - \frac{1}{CR} \tag{2.3}$$

$$CR = \frac{A}{B} \tag{2.4}$$

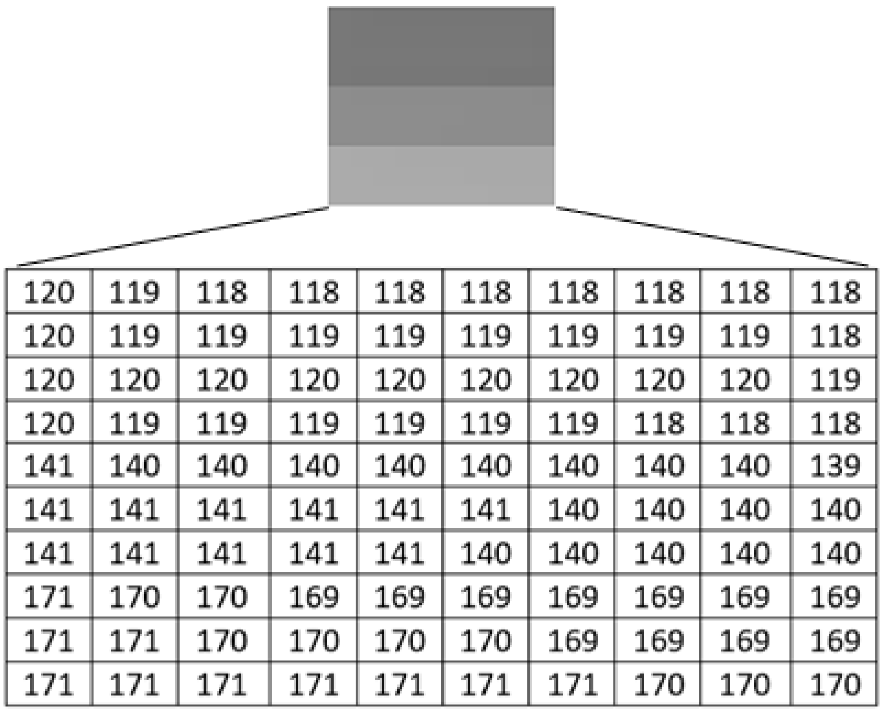
Three results can be deduced from Equations 2.3 and 2.4.

1. When A = B, CR = 1, $R_{dr} = 0$, there is no redundancy, and hence no compression.

2. When B $\ll A, CR \rightarrow infinite,\ R_{dr} \rightarrow 1$, dataset A contains the highest redundancy and the greatest compression is achieved.

3. When $B \gg A, CR \rightarrow 0,\ R_{dr} \rightarrow -infinite$, dataset A contains large memory than the original.

In digital image compression, there are three types of data redundancy: coding, inter-pixel, and psycho-visual redundancy [41, 42]. Suppose we have the image shown in Figure 2.7 with the corresponding grey levels.



120	119	118	118	118	118	118	118	118	118
120	119	119	119	119	119	119	119	119	118
120	120	120	120	120	120	120	120	120	119
120	119	119	119	119	119	119	118	118	118
141	140	140	140	140	140	140	140	140	139
141	141	141	141	141	141	140	140	140	140
141	141	141	141	141	140	140	140	140	140
171	170	170	169	169	169	169	169	169	169
171	171	170	170	170	170	169	169	169	169
171	171	171	171	171	171	171	170	170	170

Figure 2.7: An image with the corresponding pixel values

The 10x10 image shown in Figure 2.7 contains nine different values (S) (118, 119, 120, 139, 140, 141, 169, 170, 171), and for a fixed code length, each values can be coded as an 8-bit code-word, since the maximum value (171) requires a minimum of 8 bits to code. As a result, 800 bits are required to store the image. In contrast, a variable code length is based on probability, where codes of shorter length are assigned to values with higher probability. The probability of the k^{th} values is calculated using Equation 2.5, where N is the total number of values in an image. If L_k represents the length of the code-word for the values S_k, then the length of the average code-word can be calculated using Equation 2.6, where SL is the total number of different values. Table 2.1 shows the variable length coding for the image in Figure 2.7.

$$P(k) = \frac{S_k}{N} \tag{2.5}$$

Table 2.1: Variable length coding for the image in Figure 2.7

Symbol (S)	Probability (P(k))	Code-word	Code-word length (L_k)	L_kP_k
118	0.12	100	3	0.36
119	0.16	1	3	0.48
120	0.12	11	3	0.36
139	0.01	1111	4	0.04
140	0.17	0	3	0.51
141	0.12	10	3	0.36
169	0.11	101	3	0.33
170	0.09	1110	4	0.36
171	0.1	110	3	0.3
				L_{avg} = 3.1 bits

$$L_{avg} = \sum_{k=1}^{SL} L_k P(k) \tag{2.6}$$

From Table 2.1, we get approximate values of CR = 2.581 and $R_{dr} = 0.613$, and the compressed image takes 300 bits rather than 800 bits. These results show that the original image contains redundant code, and that the variable length coding has removed this redundancy [43,44].

Interpixel redundancy can be classified as spatial, spectral, and temporal redundancy. In spatial redundancy, there are correlations between neighbouring pixel values, whereas in spectral redundancy there are correlations between different spectral bands. In temporal redundancy, there are correlations between the adjacent frames of a video. Interpixel redundancy can be removed using run-length coding, the differences between adjacent pixels, predicting a pixel using various methods, thresholding, or various types of transformation techniques such as discrete Fourier transform (DFT) [44].

To remove interpixel redundancy from the image in Figure 7 using run-length coding, we code the image as follows: (120,1) (119,1) (118,8) (120,1) (119,8) (118,1) (120,9) (119,1) (120,1) (119,6) (118,3) (141,1) (140,8) (139,1) (141,6) (140,4) (141,5) (140,5) (171,1) (170,2) (169,7) (171,2) (170,4) (169,4) (171,7) (170,3), requiring 312 bits. Twelve bits are required to code each pair, and an 8-bit code word is used for the grey level, since the maximum gray level is 171. A 4-bit code word is used for the length of the grey level, since the maximum value for the length of the gray level is nine.

The main purpose of using prediction or transformation techniques can be described as follows. To create a narrow histogram, a prediction or various other types of transformation techniques can be applied to give a small value for the entropy. For example, we apply the very simple predictor given below to the image shown in Figure 2.7, where $A^{'}$ represents the predicted pixels. Figure 2.8(a) shows the histogram of the original image, and Figure 2.8(b) shows the histogram obtained after applying the predictor shown in Equation 2.7.

$$A^{'}(p,q) = A(p,q-1) - A(p,q) \tag{2.7}$$

Figure 2.8 shows that the histogram of the original image contains nine different values, of which eight have approximately the same frequency, and the highest frequency is 17. After applying the predictor, the histogram contains only five values, of which only two (0 and 1) contain 90% of the data, thus giving a better compression ratio. The interpixel redundancy of an image can be removed using one or more techniques in combination. For example, after

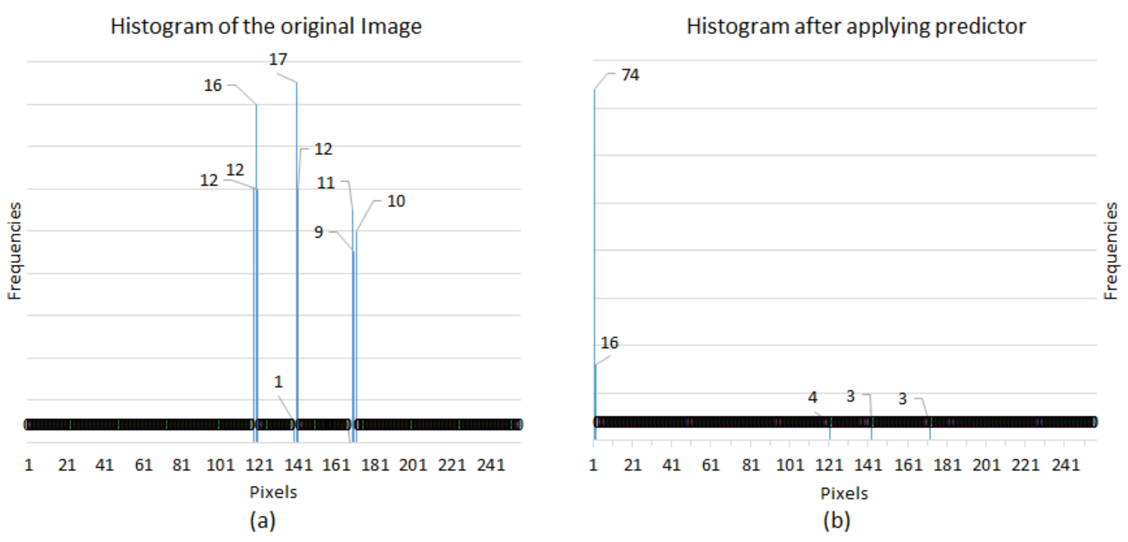
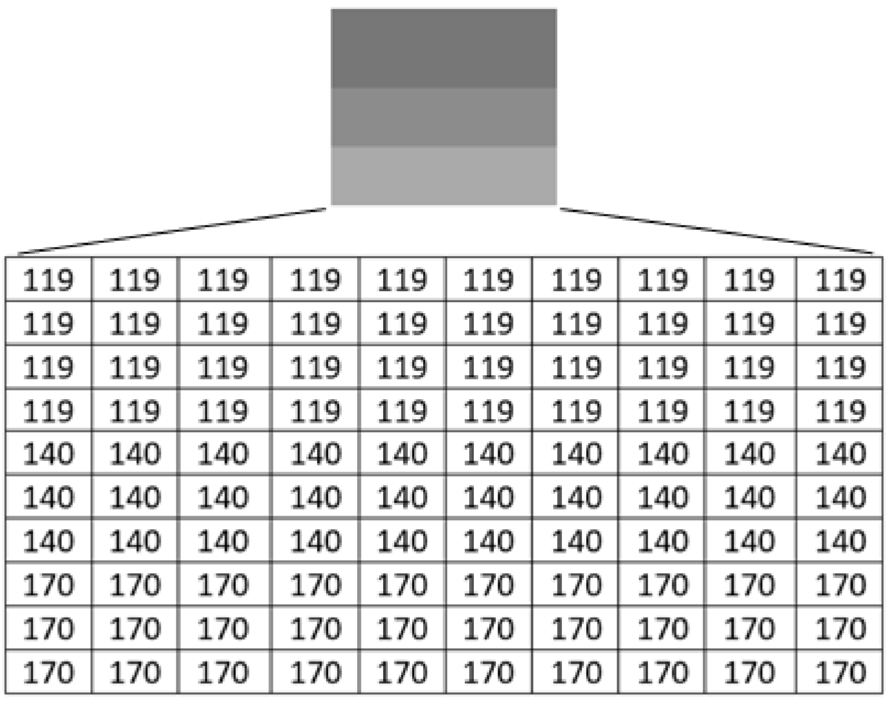


Figure 2.8: Comparison of histograms before and after application of a predictor

applying the predictor to the original image in Figure 2.7, we can apply both run-length and Huffman coding, with the result that only 189 bits are required to store the image rather than 800 bits.

Psycho-visual redundancy [45] simply reduces the grey levels of an image. The human brain responds to the most important features, such as edges and textures, rather than using all of the visual information to recognise an object. The image in Figure 2.9 shows the results of removing the redundant psycho-visual data from the image in Figure 2.7. Although there is a large difference between the two images in terms of the grey levels (Figure 2.9 contains only three grey levels rather than nine), the brain processes them similarly. Psycho-visual redundancy can therefore be used for lossy image compression.



119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	119	119	119	119	119
119	119	119	119	119	119	119	119	119	119
140	140	140	140	140	140	140	140	140	140
140	140	140	140	140	140	140	140	140	140
140	140	140	140	140	140	140	140	140	140
170	170	170	170	170	170	170	170	170	170
170	170	170	170	170	170	170	170	170	170
170	170	170	170	170	170	170	170	170	170

Figure 2.9: Psycho-visual redundant image with grey levels

The construction of an image compression technique is highly application-specific. A general block diagram of lossless image compression and decompression is shown in Figure 2.10.

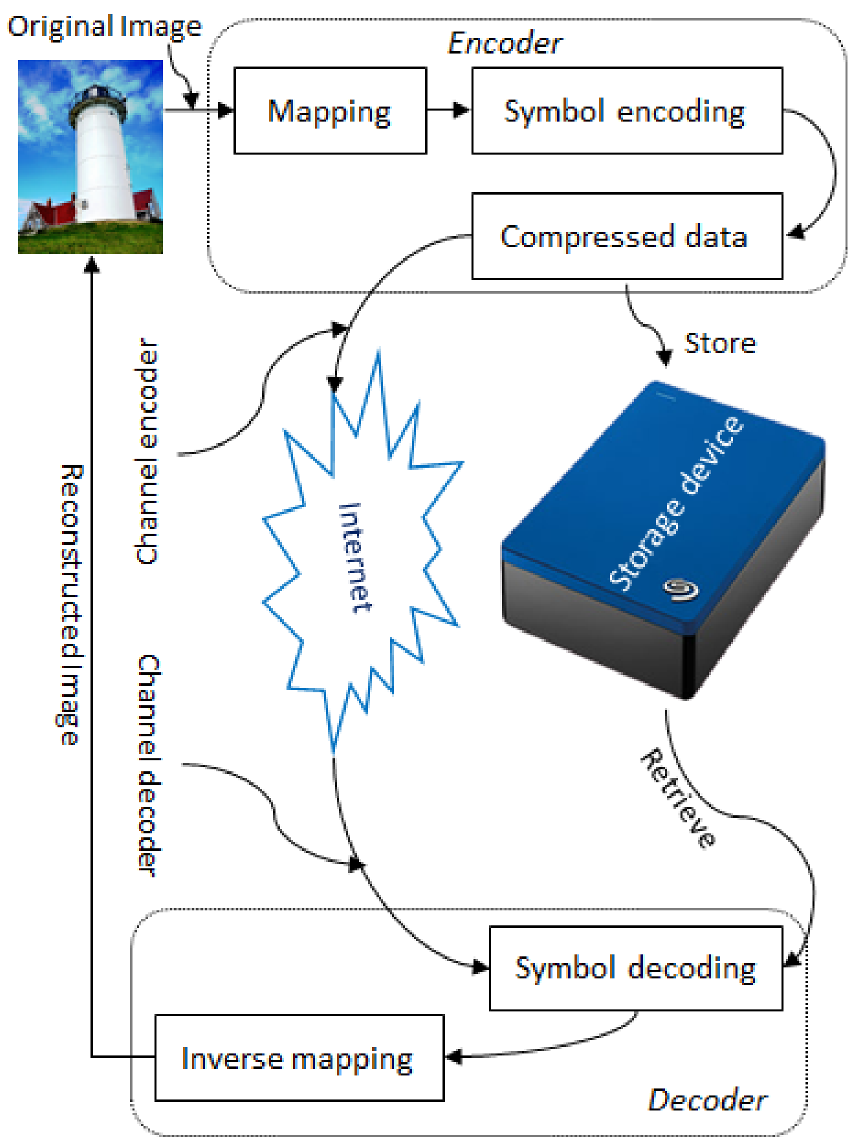


Figure 2.10: Basic block diagram of lossless image compression

The mapping shown in Figure 2.10 is used to convert an image into a non-visual form to decrease the interpixel redundancy. Run-length coding, various transformation techniques, and prediction techniques are typically applied at this stage. At the symbol encoding stage, Huffman, arithmetic and other coding methods are often used to reduce coding redundancy. The image data are highly correlated, and the mapping process is a very important way of decorrelating the data and eliminating redundant data. A better mapping process can eliminate more redundant data and give better compression. The first and most important problem in image compression is to develop or choose an optimal mapping process, while the second is to choose an optimal entropy coding technique to reduce coding redundancy [46]. In channel encoding, Hamming coding is applied to increase noise immunity, whereas in decoding, the inverse procedures are applied to give a lossless decompressed image. Quantisation, an irreversible process, removes irrelevant information by reducing the number of grey levels, and is applied between the mapping and symbol encoding stages in lossy image compression [47–49].

2.3 Measurement Standards

Measurement standards offer ways of determining the efficiency of an algorithm. The seven measurement standards are used to evaluate a lossless image compression algorithm. Three of these, the MSE, SNR, and PSNR, are used to detect the amount of error present in a reconstructed image. The MSE is often called the quantisation error variance or the mean squared deviation (MSD). It represents an approximate measurement of the distortion, and is used to determine the quality of a reconstructed image B (p, q) compared to the original A (p, q). The MSE is calculated using Equation 2.8 [50–52].

$$MSE = \frac{1}{pxq} \sum_{m=1}^{p} \sum_{n=1}^{q} [A(m,n) - B(m,n)]^2 \tag{2.8}$$

The value of the MSE is inversely proportional to image quality: a very small MSE represents a high-quality reconstructed image [53–55]. However, one problem with the use of MSE is that when the types of degradation of two images are different, a low MSE does not mean that the original and reconstructed images are almost identical. In this case, many applications use the SNR in place of the MSE. The SNR is defined in Equation 2.9 [56–58], and provides a better indication of noise.

$$SNR = 10 * log_{10} \left[\frac{\frac{1}{pxq} \sum_{m=1}^{p} \sum_{n=1}^{q} [A(m,n)]^2}{\frac{1}{pxq} \sum_{m=1}^{p} \sum_{n=1}^{q} [A(m,n) - B(m,n)]^2} \right] \tag{2.9}$$

Another criterion is the PSNR, which is used when considering the size of the error relative to the 2^{nbpp} value for an image, where nbpp is the number of bits per pixel. A higher value for the PSNR represents a higher-quality image. The PSNR is defined by Equation 2.10:

$$PSNR = 10 * log_{10} \left[\frac{(2^{nbpp} - 1)^2}{MSE} \right] \tag{2.10}$$

The CR is the ratio between the size of an uncompressed image and its compressed version. Entropy is generally estimated as the average code length of a pixel in an image; however, in reality, due to statistical interdependencies among pixels, this estimate is overoptimistic. For example, Table 1 shows that the CR is 2.581 but the estimated entropy for the same data is 3.02. Hence, the entropy-based compression ratio is 2.649. To solve this issue, Equation 2.11 is used to calculate the bits per pixel(bpp). The bpp [59] is the number of bits used to represent a pixel, i.e. the inverse of the CR. The ET and DT are the times required by an algorithm to encode and decode an image, respectively.

$$bpp = \frac{B}{A} \tag{2.11}$$

2.4 Summary

Data compression has been used for hundreds of years. The introduction of information theory and communication networks has comprehensively accelerated the area and added necessity to its use. At the beginning of this chapter, a brief explanation has been given on how to convert an image from a continuous tone image to a digital form and about the structure of various kinds of images. Following that, an image compression procedure has been shown step by step for quick understanding. Finally, The metrics used to evaluate a data compression algorithm are discussed, and their limitations are also addressed.

Chapter 3

Transformation Techniques

3.1 Introduction

A transform applied for data compression is a mathematical computation same as Fourier transform, which is executed by grouping the input data into vectors or matrices and multiplying these with a transform matrix to weight the input representations. Applying a transform technique in data compression is that the transform should decorate the representations in the input data and concentrate the energy in the first resulting transform coefficients. The energy of a matrix or vector is represented as the sum of the squares of the components $p^2+q^2+r^2+s^2+\dots$. If the energy is focused in the first coefficients, they keep the essential information and have large values. In contrast, the later coefficients have small and not-so-essential values. Compression can be obtained by heavily quantizing the small insignificant coefficients and lightly quantizing the significant coefficients or even not modifying them.

The transform coefficients can further be viewed as the frequency components of the input data. The frequency expresses how vital parts of the input data are varying between them. The initial transform coefficient describes the DC element with frequency 0, whereas the latter coefficients correspond to AC components of the input data with increasing frequencies. Transformation techniques [60, 61] is a mathematical operation that receives an input sequence and maps it into another form. There are many advantages of the transformation. For example, the transformed sequence may require less storage space and provide data compression. We can easily apply an operation on the transformed data than the original. In the field of data compression, transformation is widely used because of its decorrelation and other characteristics. There are many transformation techniques: such as discrete cosine transform (DCT), discrete wavelet transform (DWT), Burrows-Wheeler Transform (BWT), etc. Some of the transformation techniques are explained in this chapter.

3.1.1 Discrete Wavelet Transform (DWT)

Discrete Fourier Transform (DFT) decomposes an image into sinusoidal basis functions of different frequencies, where Wavelet Transform (DWT) decomposes into a set of mutually orthogonal wavelet basis functions. The main difference between DFT and DWT is that the wavelet basis functions are spatially localized. However, both transformation techniques are entirely reversible. There are many wavelet functions. The two most common are the Haar and Daubechies-4 wavelets functions. DWT decomposes a two dimensional data (like an image) into four bands: LL (left-top), HL (right-top), LH (left-bottom) and HH (right-bottom). HL and LH bands, respectively, indicate the variation along the x-axis and y-axis. In other words, HL and LH give the vertical and horizontal features of an image. HH gives the diagonal features, and LL approximates the input image, which is further decomposed. The general block diagram of DWT decomposition for an image (p x q) is shown in 3.1.

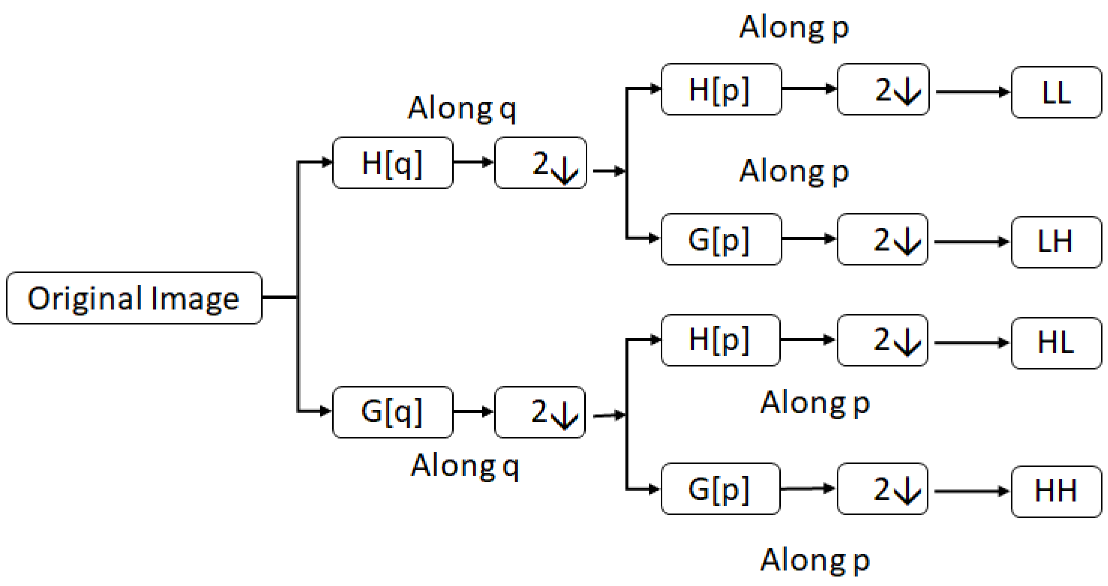


Figure 3.1: Basic block diagram of 2D wavelet transform

Haar Wavelet

Mathematically, the Haar wavelet family is a sequence of square-shaped functions, which is the most straightforward orthogonal wavelet transform. It is calculated by repeating the difference and averaging between odd and even samples of a signal. For the formation of orthonormal bases, Scaling functions play a significant role. The idea of scaling functions is most clearly explained using Haar wavelets. The Haar scaling function is defined by the following equation (3.1). It meets the normalization conditions shown in equation (3.2).

$$\phi(p) = \begin{cases} 0 & p \leq 0 \\ 1 & 0 < p \leq 1 \\ 0 & p > 1 \end{cases} \tag{3.1}$$

$$(\phi, \phi) = \int_{-\infty}^{\infty} \phi^*(p)\phi(p)dp = \int_0^1 \phi(p)dp = 1 \tag{3.2}$$

A practical example of haar wavelet transformation is given below step by step. This transform involves the forward and the reverse transform. In the forward transformation scaling and wavelet coefficients are calculated. The scaling and wavelet coefficients are computed using the following equations (3.3), and (3.4), respectively, where a and b are two adjacent pixels of an image. Transformation happens column by column and then row by row. Let consider a matrix (HM) is given in the following equation (3.5).

$$Scaling\_coefficients = \frac{a+b}{2} \tag{3.3}$$

$$Wavelet\_coefficients = \frac{a-b}{2} \tag{3.4}$$

$$HM = \begin{pmatrix} 100 & 200 & 80 & 120 \\ 40 & 60 & 180 & 220 \\ 200 & 200 & 120 & 100 \\ 300 & 100 & 20 & 40 \end{pmatrix} \tag{3.5}$$

In the first state, the Haar forward wavelet transform is applied column by column, and the following transformed matrix (3.6) is calculated from the original matrix (3.5). In the second stage, the same operation is performed row by row on the transformed matrix (3.6). Finally, we get the matrix (3.7) as a transformed matrix after one pass.

$$HM = \begin{pmatrix} 150 & 100 & -50 & -20 \\ 50 & 200 & -10 & -20 \\ 200 & 110 & 0 & 10 \\ 200 & 30 & 100 & -10 \end{pmatrix} \tag{3.6}$$

$$HM = \begin{pmatrix} 100 & 150 & -30 & -20 \\ 200 & 70 & 50 & 0 \\ 50 & -50 & -20 & 0 \\ 0 & 40 & -50 & 10 \end{pmatrix} \tag{3.7}$$

In terms of inverse Haar Wavelet Transform, the transform first goes row by row and then column by column using equations (3.8), and (3.9). The output of the first stage of the inverse transformation is the same as the matrix (3.6), and the reconstructed matrix after one pass transformation is the same as the original matrix HM.

$$X_{i,j} = a_{i,j} + a_{i+2,j} \tag{3.8}$$

$$X_{i+1,j} = a_{i,j} - a_{i+2,j} \tag{3.9}$$

Daubechies-4 Wavelet

In the frequency domain, sinusoidal functions are perfectly localized using Fourier transform, and it is difficult in the spatial domain. However, the wavelet basis is perfectly localized in both domains. In addition to this localization, we need that all basis functions be mutually normalized and orthogonal. These features are obtained through the recursion process of Daubechies wavelets formula. The scaling functions and wavelets are defined using the dilation equation (3.10), where $\phi(p)$ is known as the scaling function. The scaling function is normalized by equation (3.11). The fourth-order mother wavelet $\psi(p)$ is defined in equation (3.12) in terms of scaling function.

$$\phi(p) = \sqrt{2} \sum_{i=0}^{M-1} C_i \phi(2p - i) \tag{3.10}$$

$$\int \phi(p)\, dp = 1 \tag{3.11}$$

$$\psi(p) = \sqrt{2}\sum_{i=0}^{M-1}(-1)^i C_{M-1-i}\phi(2p-i) \tag{3.12}$$

The filter coefficients of the Daubechies (D4) are shown in the following equations (3.13-3.20) reported in [62]. And the coefficients are used to decompose a signal, where h and g represent the high and low frequency, respectively. For example, using the Daubechies discrete wavelet transform, the transformation and reconstruction of a signal (A = [2 4 5 7 2 3 1 6]) are explained below step by step.

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}} \tag{3.13}$$

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}} \tag{3.14}$$

$$h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}} \tag{3.15}$$

$$h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{3.16}$$

$$g_0 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{3.17}$$

$$g_1 = -\frac{3-\sqrt{3}}{4\sqrt{2}} \tag{3.18}$$

$$g_2 = \frac{3+\sqrt{3}}{4\sqrt{2}} \tag{3.19}$$

$$g_3 = -\frac{1+\sqrt{3}}{4\sqrt{2}} \tag{3.20}$$

This transformation uses four filter coefficients and the following matrix (3.21). The outputs are calculated using the equation (3.22), and the final encoded outputs are 4.51, -0.3, 8.33, -1.95, 2.92, -2.96, 5.44, and -1.69.

$$W = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ h_3 & -h_2 & h_1 & -h_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & h_3 & -h_2 & h_1 & -h_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & h_3 & -h_2 & h_1 & -h_0 \\ h_2 & h_3 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ h_1 & -h_0 & 0 & 0 & 0 & 0 & h_3 & -h_2 \end{pmatrix} \tag{3.21}$$

$$W(i) = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ h_3 & -h_2 & h_1 & -h_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & h_3 & -h_2 & h_1 & -h_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & h_3 & -h_2 & h_1 & -h_0 \\ h_2 & h_3 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ h_1 & -h_0 & 0 & 0 & 0 & 0 & h_3 & -h_2 \end{pmatrix} \times \begin{pmatrix} 2 \\ 4 \\ 5 \\ 7 \\ 2 \\ 3 \\ 1 \\ 6 \end{pmatrix} \tag{3.22}$$

The reconstruction process is done using the inverse wavelet transformation, which is just the W's transposed matrix. The Daubechies inverse transformation matrix is given in the following equation (3.23). Finally, the reconstructed outputs ($\overline{A}$) are calculated using the equation (3.24) reported in [63], and the decoded outputs are 1.981, 3.958, 4.992, 6.984, 1.981, 2.957, 2.018, 9.915.

$$W^T = \begin{pmatrix} h_0 & h_3 & 0 & 0 & 0 & 0 & h_2 & h_1 \\ h_1 & -h_2 & 0 & 0 & 0 & 0 & h_3 & -h_0 \\ h_2 & h_1 & h_0 & h_3 & 0 & 0 & 0 & 0 \\ h_3 & -h_0 & h_1 & -h_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_3 & 0 & 0 \\ 0 & 0 & h_3 & -h_0 & h_1 & -h_2 & 0 & 0 \\ h_2 & h_3 & 0 & 0 & h_2 & h_1 & h_0 & h_3 \\ h_1 & -h_0 & 0 & 0 & h_3 & -h_0 & h_1 & -h_2 \end{pmatrix} \tag{3.23}$$

$$\overline{A} = \begin{pmatrix} h_0 & h_3 & 0 & 0 & 0 & 0 & h_2 & h_1 \\ h_1 & -h_2 & 0 & 0 & 0 & 0 & h_3 & -h_0 \\ h_2 & h_1 & h_0 & h_3 & 0 & 0 & 0 & 0 \\ h_3 & -h_0 & h_1 & -h_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_3 & 0 & 0 \\ 0 & 0 & h_3 & -h_0 & h_1 & -h_2 & 0 & 0 \\ h_2 & h_3 & 0 & 0 & h_2 & h_1 & h_0 & h_3 \\ h_1 & -h_0 & 0 & 0 & h_3 & -h_0 & h_1 & -h_2 \end{pmatrix} \times \begin{pmatrix} 4.51 \\ -0.3 \\ 8.33 \\ -1.95 \\ 2.92 \\ -2.96 \\ 5.44 \\ -1.69 \end{pmatrix} \tag{3.24}$$

3.1.2 Burrows–Wheeler Transform (BWT)

The Burrows-Wheeler transform (BWT), also called block-sorting compression, was invented by Michael Burrows and David Wheeler in 1994 based on Wheeler transform. It rearranges a set of characters into runs of similar characters [64]. It is used for data compression techniques such as bzip2. It is entirely reversible, and no extra information is stored without the position of the last character. The transformed character set can be easily compressed by run-length coding. The pseudo-codes of the forward and inverse Burrows-Wheeler transforms are given in [65, 66].

3.2 Summary

Many techniques have been developed to perform lossless data compression. All of these techniques try to exploit some regularities in the input data to achieve coding efficiency. There are two types of techniques such as transform and spatial domain. Transform domain techniques perform a linear transformation of the input image and usually work extremely well, and spatial domain techniques exploit local inter-pixel correlation directly. The main limitation of transform techniques is their computational complexity. This chapter provided detailed explanations of several transform techniques with examples.

Chapter 4

Lossless Data Compression Techniques

4.1 Introduction

A review of several important lossless data compression algorithms is discussed in this chapter. Section 4.2 explains the entropy coding methods such as run-length, Shannon-Fano, Huffman, LZW, and Arithmetic coding. As predictive coding, Lossless JPEG, JPEG-LS, PNG, CALIC, WebP, and FLIF are presented in section 4.3. JPEG 2000, JPEG XR, and AVIF are given as transform coding in section 4.5, and LZ77 is presented in section 4.5. Finally, we conclude this chapter in section 4.6.

4.2 Entropy Coding Methods

Entropy coding is a term that refers to lossless coding technology that replaces data elements with coded representations and vice versa. An entropy coding technique generates and assigns a unique prefix-free code to each unique symbol that happens at an input.

4.2.1 Run-Length Coding

Run-length is a lossless coding procedure that stores only a mark and a count when a series of identical values appears at consecutive times. This strategy is more cost-effective instead of encoding each pixel [41, 67, 68]. In that strategy, it chooses the first pixel from an image and then connects it to the output string and counts the number of succeeding occurrences of the selected pixels. Lastly, it appends the count to the destination string. This process continues until reading the whole pixels of an image is finished. The working procedure for run-length coding for compression and decompression are given in [69].

4.2.2 Shannon-Fano Coding

Shannon-Fano coding generates a prefix code based on a set of symbols and their frequencies. In that procedure, all frequencies are arranged in descending order and then divide into two groups whose total frequencies are as close as possible to being equal reported in [70]. The encoding and decoding procedures are given below. The Shannon-Fano encoding and decoding procedures are shown in [27].

4.2.3 Huffman coding

Huffman is a lossless data compression algorithm that uses fewer bits to encode those pixels that happen more regularly. Huffman typically builds a tree based on the probabilities of an image. Finally, encoding is done using the tree [19,71,72]. The encoding and decoding procedures of Huffman coding are given in [27].

4.2.4 Lempel–Ziv–Welch (LZW) Coding

Lempel–Ziv–Welch (LZW) is generally used for lossless text compression, invented by Abraham Lempel, Jacob Ziv, and Terry Welch. This strategy is easy to implement and broadly applied for Unix file compression, published in 1984 as an updated version of LZ78. It encodes a sequence of characters with a unique code using a table-based lookup algorithm. In this algorithm, the first 256 8-bit code, 0-255, is inserted into a table as an initial entry because an image contains 0–255 distinct pixels. The encoded codes come from 256 to 4095, which will be embedded into the table's bottom [73, 74]. This algorithm works better in text compression and provides most noticeably a terrible outcome for another sort of compression. The encoding procedure of the algorithm is shown in [27].

The Lempel–Ziv–Welch (LZW) decoding procedure uses the same initial dictionary used in the encoding step, and decoding is done using the procedures shown in [27].

4.2.5 Arithmetic Coding

Arithmetic coding is a lossless data compression procedure where a set of symbols is presented using a fixed number of bits reported in [75–80]. It uses a different approach than Huffman coding, and it does not require an integer number of bits per symbol and hence works properly in situations where Huffman coding struggles. The fundamental concept of arithmetic coding is very easy. Arithmetic coding connects sequences of symbols with various sub-intervals of [0, 1). The width of a sub-interval is proportional to the probability of the similar sequence of symbols, and the arithmetic code of a sequence of symbols is a floating-point number in the corresponding interval [81–84]. It takes likelihood data from a dataset and applies the procedures shown in [27], where N and CF indicate the number and cumulative frequency. UL, LL, LUL, and LLL indicate upper, lower, last upper, and the last lower limit of the current range, respectively. The tag value is calculated using Equation 4.1.

$$tag = \frac{LLL + LUL}{2}. \tag{4.1}$$

The decoding procedure of arithmetic coding receives tag, symbols, and corresponding probabilities, and the tag is converted into its floating-point number and follows the the methodology shown in [27] for decoding. For decompression, if the tag is in between in any range, then the range's symbol is taken as the decoded value. The range (r) and Newtag (NT) are calculated using Equations 4.2 and 4.3, respectively.

$$r = (UL - LL), \tag{4.2}$$

$$NT = \frac{tag - LL}{r}. \tag{4.3}$$

4.3 Predictive Coding

Predictive coding is a technique that predicts P_n using a predictor for the current pixel X_n from the previous N number of pixels X_{N+n}, and calculate the prediction errors E_n between the current pixel's X_n and the predicted values P_n. The main advantage of this technique is that the prediction errors considerably decrease statistical dependencies between adjacent pixels. Finally, an entropy coding technique is used to encode the prediction errors. The general block diagram of the prediction technique for data compression is shown in 4.1.

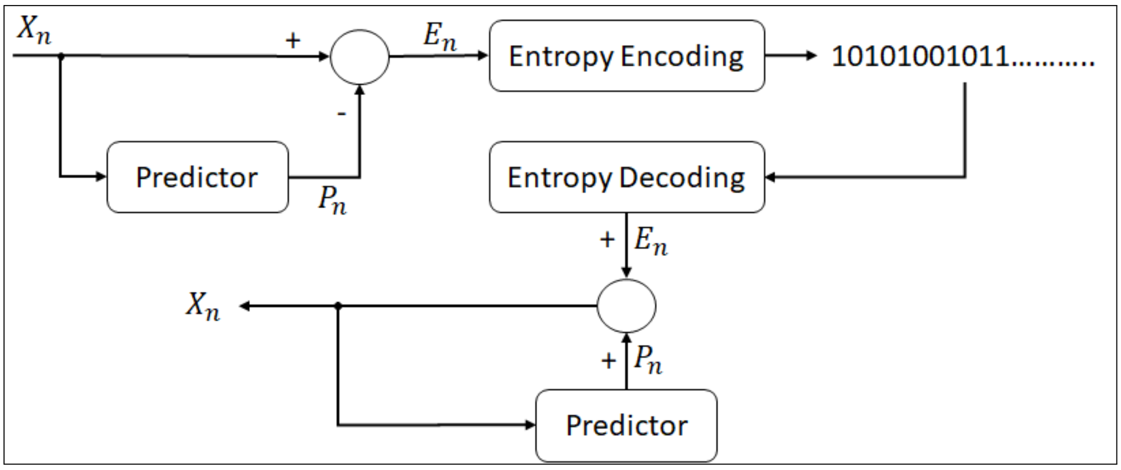
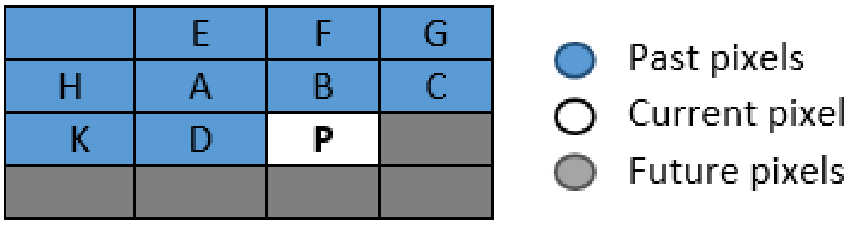


Figure 4.1: General Block Diagram of the Predictive Coding for Lossless Data Compression

4.3.1 Lossless JPEG

The Joint Photographic Experts Group (JPEG) format is a DCT-based lossy image compression technique, whereas lossless JPEG is predictive. Lossless JPEG uses the 2D differential pulse code modulation (DPCM) scheme [85], and predicts a value ($\overline{P}$) for the current pixel (**P**) based on up to three neighbouring pixels (A, B, D). The causal template used to predict a value is shown in Table 4.1. If two pixels (B, D) from Table 4.1 are considered in this prediction, and then the predicted value ($\overline{P}$) and prediction error ($\overline{PE}$) are calculated using Equations (4.4) and (4.5), respectively.

Table 4.1: Causal template



🟦	E	F	G
H	A	B	C
K	D	**P**	▨
▨	▨	▨	▨

🔵	Past pixels
○	Current pixel
●	Future pixels

$$\overline{P} = \frac{D + B}{2} \tag{4.4}$$

$$\overline{PE} = P - \overline{P} \tag{4.5}$$

As a result, the prediction errors remain close to zero, and very large positive or negative errors are not commonly seen. The error distribution therefore looks almost like a Gaussian normal distribution. Finally, Huffman or arithmetic coding is used to code the prediction errors. Table 4.2 shows the predictor used in the lossless JPEG format, based on three neighbouring pixels (A, B, D). In lossy image compression, three types of degradation typically occur and should be taken into account when designing a DPCM quantiser: granularity, slope overload, and edge-busyness [86]. DPCM is most sensitive to channel noise.

A real image usually has a nonlinear structure, and the DPCM uses a linear predictor, This is why problems can occur. This gave rise to the need to develop a perfect nonlinear predictor. One of the most widely used nonlinear predictors is the median edge detector (MED) which is used by JPEG-LS to address these drawbacks.

Table 4.2: Predictor for Lossless JPEG

Mode	Predictor
0	No prediction
1	D
2	B
3	A
4	D+B-A
5	D+(B-A)/2
6	B+(D-A)/2
7	(D+B)/2

4.3.2 Joint Photographic Experts Group-Lossless Standard (JPEG-LS)

JPEG-LS was designed based on LOCO-I (Low Complexity Lossless Compression for Images) [1, 87], and a standard was eventually introduced in 1999 after a great deal of development [88–90]. JPEG-LS improves the context modelling and encoding stages by applying the same concept as lossless JPEG. Though the discovery of arithmetic codes [91, 92] conceptually separates the stages, the separation process becomes less clean under low-complexity coding constraints, due to the use of an arithmetic coder [93]. In context modelling, the number of parameters is an important issue, and must be reduced to avoid context dilution. The number of parameters depends entirely on the number of context. A two-sided geometric distribution (TSGD) model is assumed for the prediction residuals to reduce the number of parameters. The selection of a TSGD model is a significant issue in a low-complexity framework, since a better model needs only very simple coding. Merhav et al. [94] showed that adaptive symbols can be used in a scheme such as Golomb coding, rather than more complex arithmetic coding, since the structure of Golomb codes provides a simple calculation without requiring the storage of code tables. Hence, JPEG-LS uses Golomb codes at this stage. Lossless JPEG cannot provide an optimal CR, because it cannot de-correlate by first order entropy of their prediction residuals. In contrast, JPEG-LS can achieve good decorrelation and provide better compression performance [95, 96]. A general block diagram for JPEG-LS is shown in Figure 4.2.

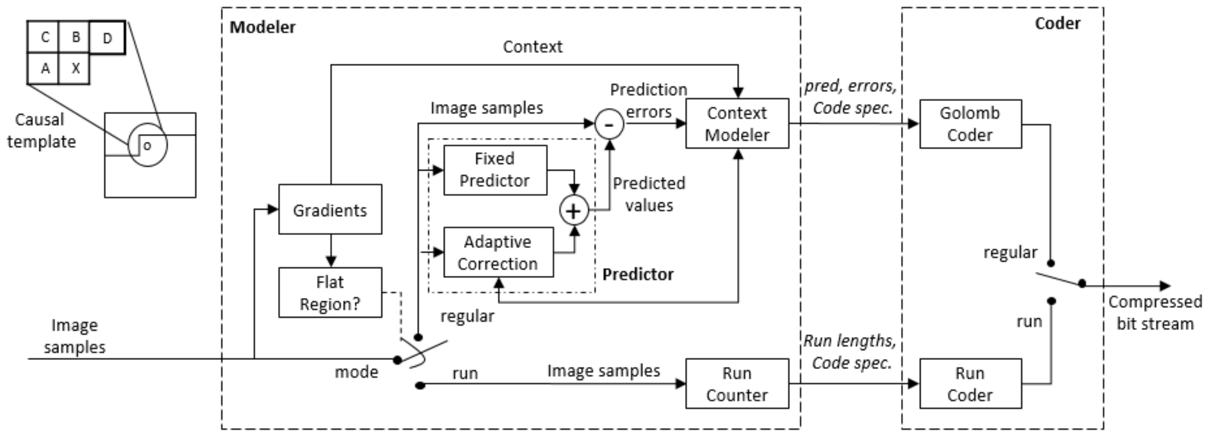


Figure 4.2: Block diagram for the JPEG-LS encoder [1]

The prediction or decorrelation process of JPEG-LS is completely different from that in lossless JPEG. As shown in Table 4.1, the LOCO-I or MED predictor used by JPEG-LS [96] predicts a value $(\overline{P})$ according to Equation (4.6).

$$\overline{P} = \begin{cases} min(D,B), & \text{if } C \geq max(D,B) \\ max(D,B), & \text{if } C \leq min(D,B) \\ D+B-A, & \text{otherwise.} \end{cases} \tag{4.6}$$

4.3.3 Portable Network Graphics (PNG)

Portable Network Graphics (PNG) [97–99], a lossless still image compression scheme, is an improved and patent-free replacement of the Graphics Interchange Format (GIF). It is also a technique that uses prediction and entropy coding. The deflate algorithm, a combination of LZ77 and Huffman coding, is used as the entropy coding technique. PNG uses five types of filter for prediction [100], as shown in Table 4.3 (based on Table 4.1).

Table 4.3: Predictor for PNG-based image compression

Type byte	Filter name	Prediction
0	None	Zero
1	Sub	D
2	Up	B
3	Average	The rounded mean of D and B
4	Paeth [55]	One of D, B, or A (whichever is closest to P = D+B-A)

4.3.4 Context-Based, Adaptive, Lossless Image Codec (CALIC)

Context-based, adaptive, lossless image codec (CALIC) is a lossless image compression technique that uses a more complex predictor (gradient-adjusted predictor, GAP) than lossless JPEG, PNG, and JPEG-LS. GAP provides better modelling than MED by classifying the edges of an image as either strong, normal, or weak. Although CALIC provides more compression than JPEG-LS and better modelling, it is computationally expensive. As shown in Table 4.1, Wu [101] used the local horizontal (Gh) and vertical (Gv) image gradients (Equations (4.7) and (4.8) to predict a value $(\overline{P})$ (Equation (4.9) for the current pixel (P) using Equation (4.10). At the coding stage, CALIC uses either Huffman coding or arithmetic coding; the latter provides more compression, but takes more time for encoding and decoding since arithmetic coding is more complex. The encoding and decoding methods in CALIC follow a raster scan order, with a single pass of an image. There are two modes of operation, binary and continuous tone. If the current locality of an original image has a maximum of two distinct intensity values, binary mode is used; otherwise, continuous tone mode is used. The continuous tone approach has four components: prediction, context selection and quantisation, context modelling of prediction errors, and entropy coding of the prediction errors. The mode is selected automatically, and no additional information is required [102].

$$Gh = |D-K| + |B-A| + |C-B| \tag{4.7}$$

$$Gv = |D-A| + |B-F| + |C-G| \tag{4.8}$$

$$\overline{P} = \begin{cases} D, & \text{if}(Gv - Gh > 80); \quad \text{Sharp horizontal edge} \\ \frac{M+P}{2}, & \text{if}(Gv - Gh > 32); \quad \text{Horizontal edge} \\ \frac{3*M+P}{4}, & \text{if}(Gv - Gh > 8); \quad \text{Weak horizontal edge} \\ B, & \text{if}(Gv - Gh < -80); \quad \text{Sharp vertical edge} \\ \frac{M+B}{2}, & \text{if}(Gv - Gh < -32); \quad \text{Vertical edge} \\ \frac{3*M+B}{4}, & \text{if}(Gv - Gh < -8); \quad \text{Weak vertical edge} \end{cases} \tag{4.9}$$

$$M = \frac{D+B}{2} + \frac{C-A}{4} \tag{4.10}$$

4.3.5 WebP

In 2010, Google introduced WebP based on VP8 [103, 104]. It is now one of the most successful image formats and supports both lossless and lossy compression. WebP predicts each block based on three neighbor blocks, and blocks are predicted in four modes: horizontal, vertical, DC, and TrueMotion [105, 106].

The lossy procedure of WebP is developed based on the intra-frame coding of the VP8 video format reported in [103, 107]. When WebP performs lossy compression, non-predicted blocks and misspredicted data are compressed in a sub-block of 4x4 pixels using a Walsh-Hadamard transform or a discrete cosine transform (DCT). Rounding errors is avoided with fixed-point arithmetic in this transform. Finally, an entropy coding technique is applied to compress the output. A clear explanation of the encoding procedure of WebP has been given in [108]. WebP supports parallel decoding.

Though WebP provides better compression than JPEG and PNG, only some browsers support WebP. Also, AVIF and JPEG-LS are developed to supersede WebP. Another disadvantage is that lossless WebP does not support progressive decoding [104, 105].

4.3.6 Free Lossless Image Format (FLIF)

Free Lossless Image Format (FLIF) is one of the best lossless image formats and provides better performance than the state-of-the-art techniques in terms of compression ratio. Many image compression techniques (e.g. PNG) support progressive decoding that can show an image without downloading the whole image. In this stage, FLIF is better as it uses progressive interlacing that is an improved version of the progressive decoding of PNG. FLIF is developed based on MANIAC (Meta-Adaptive Near-zero Integer Arithmetic Coding), a variant of CABAC (context-adaptive binary arithmetic coding. The detailed coding explanation of FLIF is given in [109, 110]. One of the main advantages of FLIF is that it is responsive by design. As a result, users can use it as per their needs. FLIF provides excellent performances on any kind of image [111, 112]. In [109], Jon et al. show that JPEG and PNG work well on photographs and drawings images, respectively, and there is no single algorithm that works well on all types of images. However, they finally conclude that only FLIF works better on any kinds of image. FLIF has many limitations such as no browser still supports FLIF [113] and takes more time for encoding and decoding an image.

4.4 Transform Coding

Transform coding is a sort of data compression technique that linearly transforms an image to concentrate as much energy into as few transform coefficients as possible. This type of transformation provides no compression but allows the many small coefficients to be approximated by zero. These techniques are excellent as high-frequency components are reduced and efficiently adapted to take advantage of the human visual system's frequency response features. In terms of lossless compression, the transformation is perfectly reversible. A better quantization technique is applied for lossy compression, resulting in a lower quality copy of the original data.

In these types of transformation techniques, computational complexity is the main limitation. A two-dimensional image's transformation takes approximately $nlog_2n$ additions and multiplications per pixel. There are many techniques used to reduce the complexity. The first is the use of separable transforms that decrease computation by doing all the horizontal and then the vertical transformations of an image. This technique reduces the calculation to log_2n. The second technique breaks an image into blocks and then transforms each of those blocks independently. Since the forward and inverse transform are both complex, the encoding time is approximately equal to decoding time. These kind of image transformation techniques are discrete wavelet transform (DWT), discrete cosine transform (DCT), etc. A general block diagram of transform coding is shown in 4.3.

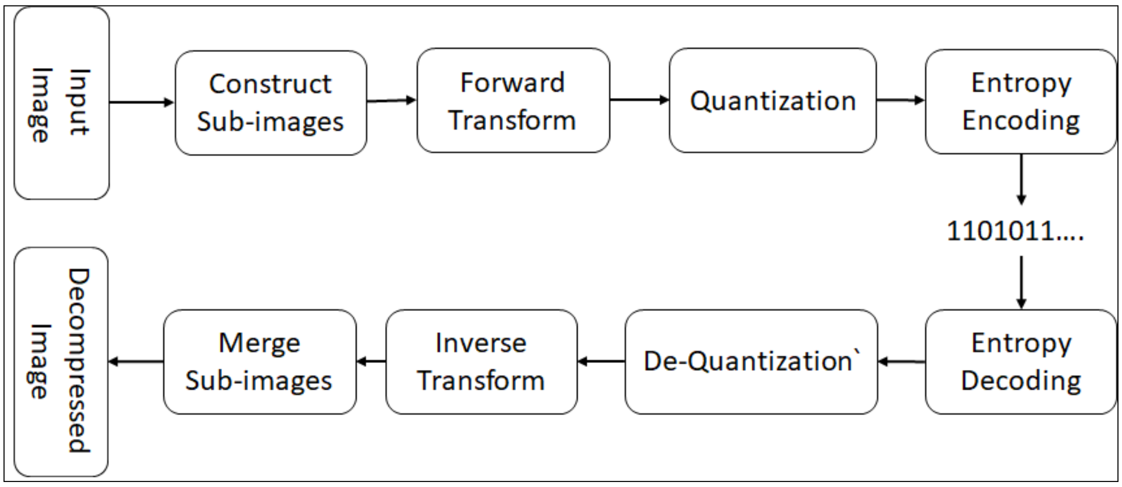


Figure 4.3: A General Block Diagram of Transform Coding

4.4.1 JPEG 2000

JPEG 2000 is an extension complement of the JPEG standard [114], and is a wavelet-based still image compression technique [115–118] with certain new functionalities. It provides better compression than JPEG [119]. The development of JPEG 2000 began in 1997, and become an international standard [120] in 2000. It can be used in lossless or lossy compression within a single architecture. Most image or video compression standards divide an image or a video frame into square blocks, to be processed independently. For example, JPEG uses the Discrete Cosine Transform (DCT) to split an image into a set of 8x8 square blocks for transformation. As a result of this processing, extraneous blocking artefacts arise during the quantisation of the DCT coefficients at high CR, producing visually perceptible faults in the image [2, 121–123]. In contrast, JPEG 2000 transforms an image as a whole using a discrete wavelet transformation (DWT), and this addresses the issue. One of the most significant advantages of using JPEG 2000 is that different parts of the same image can be saved with different levels of quality if necessary [124]. Another advantage of using DWT is that it transforms an image into a set of wavelets, which are easier to store than pixel blocks [125, 126]. JPEG 2000 is also scalable, meaning that a code stream can be truncated at any point. In this case, the image can be constructed but the resolution may be poor if many bits are omitted. JPEG 2000 has two major limitations: it produces ringing artifacts near the edges of an image, and is computationally more expensive [126]. A general block diagram for the JPEG 2000 encoding technique is shown in Figure 4.4.

Initially, an image is transformed into the YUV colour space rather than YCbCr for lossless

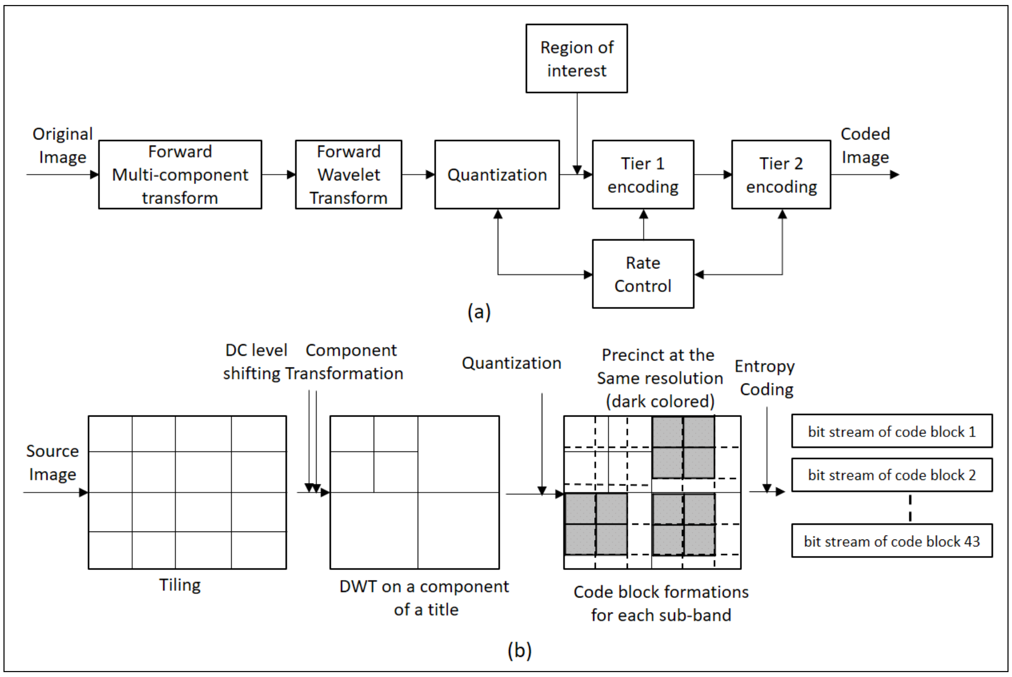

Region of interest
Original Image
Forward Multi-component transform
Forward Wavelet Transform
Quantization
Tier 1 encoding
Tier 2 encoding
Coded Image
Rate Control
(a)
DC level shifting
Component Transformation
Quantization
Precinct at the Same resolution (dark colored)
Entropy Coding
Source Image
bit stream of code block 1
bit stream of code block 2
bit stream of code block 43
Tiling
DWT on a component of a title
Code block formations for each sub-band
(b)

Figure 4.4: (a) JPEG 2000 encoder; (b) dataflow [2]

JPGE2000 compression, since YCbCr is irreversible and YUV is completely reversible. The transformed image is then split into a set of tiles. Although the tiles may be of any size, all the tiles in an image are the same size. The main advantage of dividing an image into tiles is that the decoder requires very little memory for image decoding. In this tiling process, the image quality can be decreased for low PSNR and the same blocking artifacts like JPEG can arise when more tiles are created. The LeGall-Tabatabai (LGT) 5/3 wavelet transform is then used to decompose each tile for lossless coding [124, 125], while the CDF 9/7 wavelet transform is used for lossy compression [125].

Quantisation is carried out in lossy compression, but not in lossless compression. The outcome of the transformation process is the sub-band collection and the sub-bands are further split into code blocks, which are then coded using the embedded block coding with optimal truncation (EBCOT) process, in which the most significant bits are coded first. All bit planes of the code blocks are perfectly stored and coded, and a context-driven binary arithmetic coder is applied as an entropy coder independently to each code block. In lossy compression, some bit planes are dropped. While maintaining the same quality, JPEG 2000 provides about 20% more compression than JPEG and works better for larger images.

After the DWT transformation of each tile, we obtain four parts: the top left image with lower resolution, the top right image with higher vertical resolution, the bottom left image with lower vertical resolution, and the bottom right image with higher resolution in both directions. This decomposition process is known as dyadic [114], and is illustrated in Figure 4.5 based on a real image, where the entire image is considered as a single tile.

4.4.2 JPEG XR (JPEG extended range)

Like JPEG, JPEG Extended Range (JPEG XR) is a still image compression technique that was developed based on HD photo technology [127, 128]. The main aim of the design of JPEG XR was to achieve better compression performance with limited computational resources [129], since many applications require a high number of colours. JPEG XR can represent 2.8×10^{14} colours, compared to only 16,777,216 for JPEG. While JPEG-LS, CALIC, and JPEG 2000 use MED, GAP and DWT, respectively, for compression, JPEG XR uses a lifting-based reversible

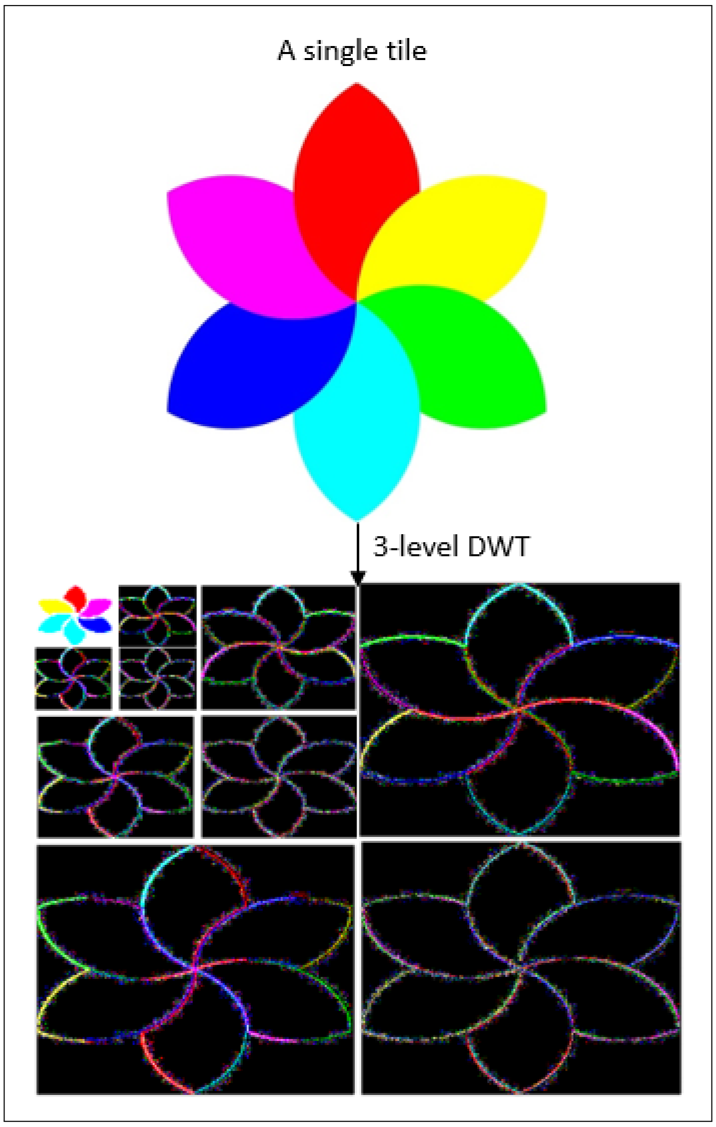


Figure 4.5: Dyadic decomposition of a single tile

hierarchical lapped biorthogonal transform (LBT). The two main advantages of this transformation are that both encoding and decoding require relatively few calculations, and are completely reversible. Two operations are carried out in this transformation: a photo core transform (PCT), which employs a lifting scheme, and a photo overlap transform (POT) [130]. Similar to DCT, PCT is a 4x4 wavelet-like multi-resolution hierarchical transformation within a 16×16 macroblock. This transformation improves the image compression performance [130]. POT is performed before PCT to reduce blocking artifacts at low bitrates. Another advantage of using POT is that it reduces the ET and DT at high bitrates [131]. At the quantisation stage, a flexible coefficient quantisation approach based on the human visual system is used that is controlled by a quantisation factor (QF), where QF varies depending on the colour channels, frequency bands, and spatial regions of the image. It should be noted that quantisation is done only for lossy compression. An inter-block coefficient prediction technique is also implemented to remove inter-block redundancy [127]. Finally, JPEG XR uses adaptive Huffman coding as an entropy coding technique. JPEG XR also allows for image tiling in the same way as JPEG 2000, meaning that the decoding of each block can be done independently. JPEG XR permits multiple colour conversions, and uses the YCbCr colour space for images with 8 bits per sample and the YCoCg color space for RGB images. It also supports the CMYK colour model [126].

General block diagrams for JPEG XR encoding and decoding are shown in Figures 4.6 and 4.7, respectively.

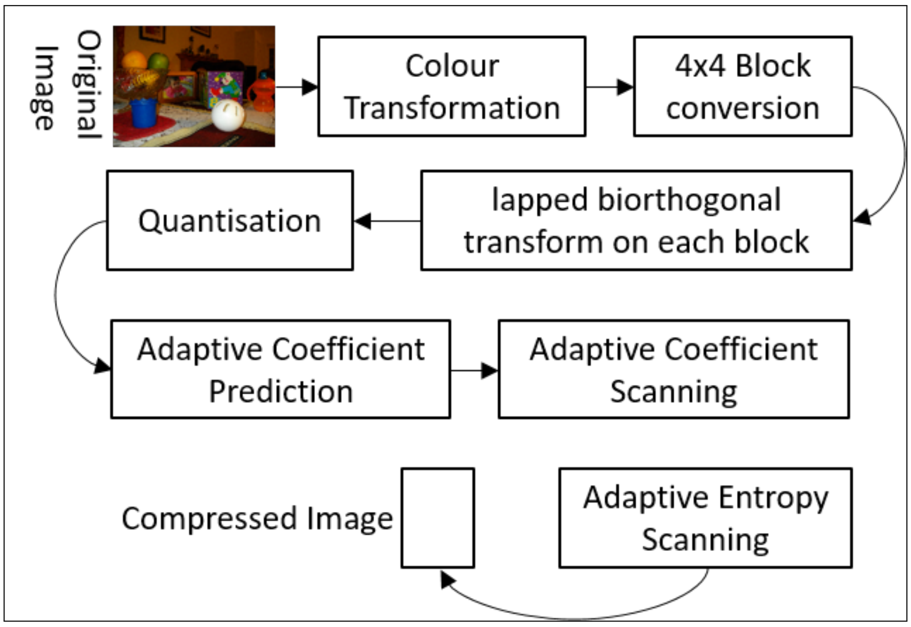

Original Image
Colour Transformation
4x4 Block conversion
lapped biorthogonal transform on each block
Quantisation
Adaptive Coefficient Prediction
Adaptive Coefficient Scanning
Adaptive Entropy Scanning
Compressed Image

Figure 4.6: JPEG XR encoder [84]

4.4.3 AV1 Image File Format (AVIF)

AOMedia Video 1 (AV1), developed in 2015 for video transmission, is a royalty-free video coding format [132], and AV1 Image File Format (AVIF) is derived from AV1 and uses the same technique for image compression. It supports both lossless and lossy compression. AV1 uses a block-based frequency transform and incorporates some new features based on Google's VP9 [133]. As a result, the AV1's encoder gets more alternatives to allow better adaptation to various kinds of input and outperforms H.264 [134]. The detailed coding procedure of AV1 is given in [135, 136]. AVIF and HEIC provide almost similar compression. HEIC is patent-encumbered H.265 format and illegal to use without getting patent licenses. On the other hand, AVIF is free to use. There are two biggest problems in AVIF. It's very slow for encoding and decoding an image, and does not support progressive rendering. AVIF provides many advantages over WebP. For example, it provides a smaller sized image and a more quality image, and supports multi-channel [137]. A detailed encoding procedure of AV1 is shown in Figure 4.8 [138].

4.5 LZ77 Algorithm

LZ77 stands for Lempel-Ziv compression method developed in 1977, a lossless data compression algorithm, developed by Abraham Lempel and Jacob Ziv in 1977 [74] and 1978 [139]. It maintains a sliding window during data compression and encodes from a sliding window over previously seen characters. But, it always starts at the beginning of the input during decoding. The pseudo-code of LZ77 is shown in [31].

4.6 Summary

Many techniques are used to compress data losslessly. They are classified into three classes: entropy coding, predictive coding, and transform coding. This chapter explained in detail

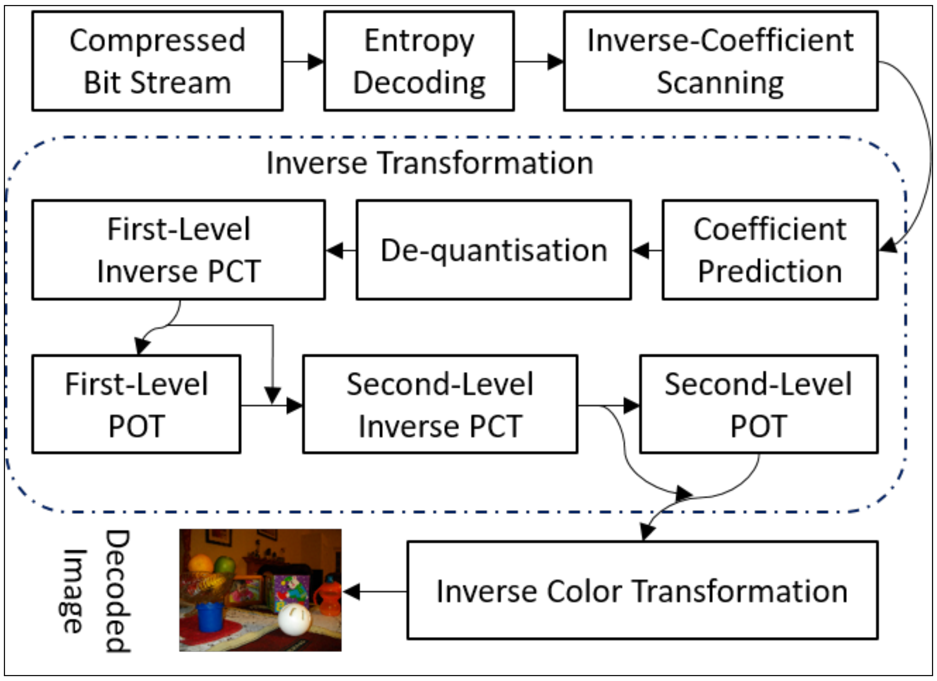


Figure 4.7: JPEG XR decoder [80]

the most widely used lossless data compression standards from each category. Their coding schemes and limitations were described in detail.

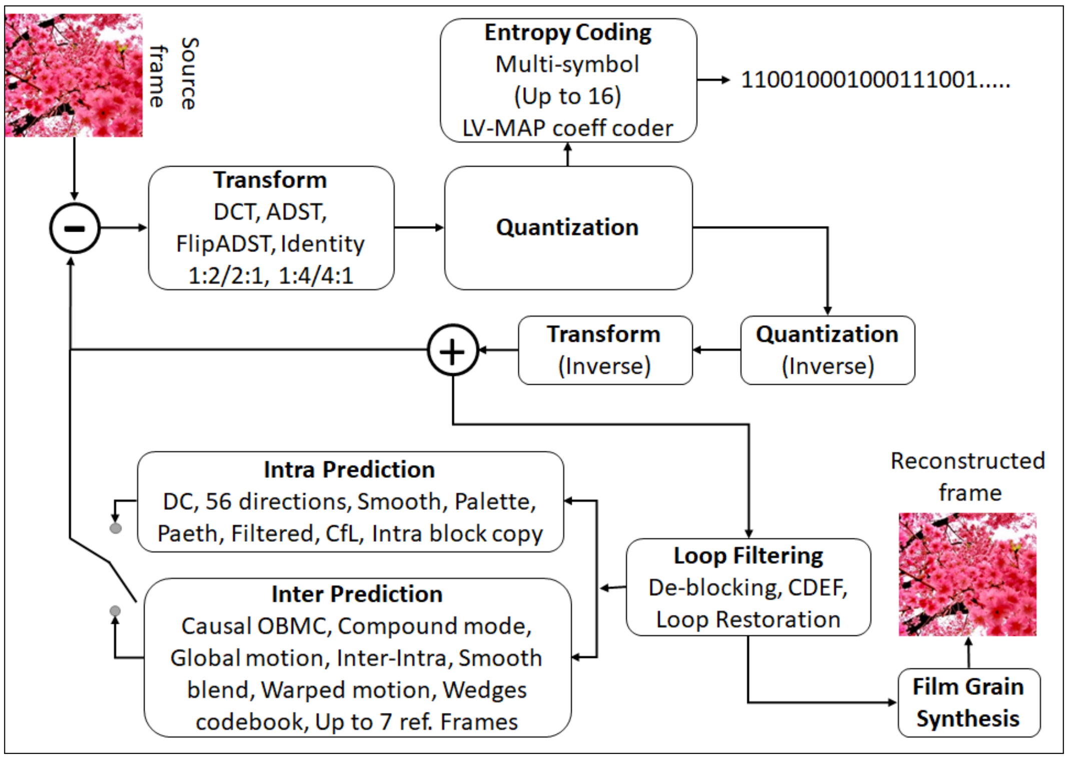


Figure 4.8: AV1 encoder

Chapter 5

Entropy coding techniques: A Survey

Modern daily life activities result in a massive amount of data, which creates a big challenge for storing and communicating them. For example, hospitals produce a considerable amount of data daily, making it a significant challenge to store it in limited storage or to communicate them through the restricted bandwidth over the Internet. Therefore, there is an increasing demand for more research in data compression and communication theory to deal with such challenges. Such research responds to the requirements of data transmission at high speed over networks. Every advanced algorithm compresses an image in two steps. At first, a prediction or transformation method is applied to an original image and then encoded using an entropy coding technique. Some of the entropy coding techniques may provide better compression than the other, while the other method may take less encoding and decoding time. Some algorithms are very sensitive to noise. Thus, choosing an entropy coding strategy for lossless data compression is a big challenge. This study focuses on a deep analysis of the most common entropy coding techniques with a common numeric example for a clear comparison. Finally, we propose the best entropy coding technique based on the analysis.

5.1 Introduction

The utilization of the computer in modernized activities is increasing virtually everywhere. As a result, sending a plethora of data, especially images and videos, over the cyber world is the most challenging issue because of circumscribed bandwidth and storage capacity; and it is time-consuming and costly, as reported in [140]. For instance, a conventional movie camera customarily uses 24 frames per second. However, current video standards sanction 120, 240, or 300 frames per second. Video is a series of still images or frames passed per second, and a color image contains three panels: red, green, and blue. Suppose you would like to send or store a three-hour color movie file of 1200×1200 dimension, and 50 frames are passed in every second. It takes approximately $(1200 \times 1200 \times 3 \times 84 \times 50 \times 10{,}800)$ bits = 17,797,851.5625 Megabits = 2172.5893 gigabytes storage if a pixel is coded in 8 bits, which is a sizably voluminous challenge to store in a computer or send over the cyber world. Here, 3 is the number of channels of a color image: R, G, and B, and 10,800 is the total number of seconds. Additionally, the medium of transmission and latency are two significant issues for data transmission. If the video file is sent over a medium of 100 Mbps, approximately (17,797,851.5625 Megabits)/100 = 177,978.5156 s = 49.4385 h is required because the medium can send 100 Megabits per second. For these reasons, compression is needed, and it is a paramount way to represent an image with fewer bits keeping its quality. An immensely colossal volume of data can be sent through an inhibited bandwidth at high speed over the cyber world reported in [47, 141]. The general block diagram of an image compression procedure is shown in Figure 5.1.

There are many image compression techniques. An image compression technique is verbally expressed to be the best when it contains less average code length, encoding, and decoding times and provides more compression ratio. Image compression algorithms are extensively applied in medical imaging, computer communication, military communication via radar, teleconferencing, magnetic resonance imaging (MRI), broadcast television, and satellite images reported in [142]. Some applications require high-quality visual information, and others need less quality, reported in [143, 144].

From this perspective, compression is divided into two types: lossless and lossy. All pristine data are recuperated correctly from an encoded data set in lossless, whereas the lossy technique retrieves virtually all data sempiternally, eliminating categorical information, especially redundant information reported in [87, 145]. Lossless is mainly utilized in facsimile transmissions of bitonal images, ZIP file format, digital medical imagery, internet telephony, and streaming video files reported in [146].

The foremost intention of implementing a compression algorithm is to diminish redundant data reported in [147]. Run-length coding, for example, is a lossless procedure where a set of the same consecutive pixels (runs of data) are preserved as a single value and a count stated in [41,82]. But, long runs of data do not exist in authentic images mentioned in [67,148], which is the main difficulty of run-length coding. Article [68] shows that a chain code binarization with run-length, and LZ77 provides a more satisfactory result than the traditional run-length technique from a compression ratio perspective. The authors in [149] show a different way of compression utilizing a bit series of a bit plane and demonstrate that it provides a better result than conventional run-length coding.

The entropy encoding techniques are proposed to solve the difficulties of a run-length algorithm. Entropy coding style encodes source symbols of an image with code words of different lengths. There are some well-recognized entropy coding methods: Shannon–Fano, Huffman, and arithmetic coding. The first entropy coding technique is Shannon–Fano, which gives a better result than run-length reported in [150]. The authors in [151] show that Shannon–Fano coding provides 30.64% and 36.51% better results for image and text compression, respectively, compared to run-length coding. However, Nelson et al. stated in [152] that Shannon–Fano sometimes generates two different codes for the same symbol and does not ascertain optimal codes, which are the algorithm's two main problems. From this perspective, Shannon–Fano coding is an inefficient data compression technique reported in [41, 82].

Huffman is another entropy coding algorithm that solves the difficulties of Shannon–Fano reported in [153,154]. In that technique, pixels that are happening more frequently are encoded, utilizing fewer bits shown in [155, 156]. Huffman coding is a good compression technique. Rufai et al. proposed a singular value decomposition (SVD) and Huffman coding-based image compression procedure [157]. SVD is used to decompose an image first, and the rank is reduced, ignoring some lower singular values. Lastly, the processed representation is coded by Huffman coding, which shows a better result than JPEG2000 for lossy compression. In [158], three algorithms: Huffman, fractal algorithm, and Discrete Wavelet Transform (DWT) coding, have been implemented and compared to show the best coding procedure. It indicates that Huffman works better to reduce redundant data, and DWT improves the quality of a compressed image, whereas the fractal provides a better compression ratio. The main problem of Huffman coding is that it is very sensitive to noise. It can not reconstruct an image perfectly from an encoded image if any changes happen [72].

Another lossless entropy method is arithmetic coding, which gives a short average code compared to Huffman coding reported in [78]. In [159], Masmoudi et al. proposed a modified arithmetic coding technique that encodes an image from top to bottom block-row wise and block by block from left to right instead of pixel by pixel using a statistical model. The precise probability between the current and its neighboring block are calculated by reducing the Kullback–Leibler gap. As a result, around 15.5% and 16.4% bitrates are decremented for static and adaptive order sequentially. A block-predicated lossless compression has been proposed using adaptive arithmetic coding and finite mixture models, reported in [24]. Here, an image is partitioned into non-overlapping blocks and encoded every block individually utilizing arithmetic coding. This algorithm provides 9.7% better results than JPEG-LS reported in [87, 160] when the work is done in a predicted error domain instead of a pixel domain. Articles [76, 77] state that arithmetic coding provides a better compression ratio. But, it takes so much time that it is virtually unutilizable for dynamic compression. Also, its use is restricted by patent. On the other hand, though Huffman coding provides marginally less compression, it utilizes significantly less time to encode an image than arithmetic coding. That's why it is suitable for dynamic compression reported in [78, 79]. Furthermore, an image encoded by arithmetic coding can corrupt the entire image for a single bit error because it has very impecunious error resistance reported in [80, 81]. Contiguous to, the primary inhibition of entropy coding is that it increments the complexity of CPU stated in [82, 83].

LZW (Lempel–Ziv–Welch) is a dictionary predicated compression technique that reads a sequence of pixels and then groups the pixels into strings. Lastly, the strings are converted into codes. In that technique, a code table with 4096 common entries is utilized, and the fixed codes 0–255 are assigned first in a table as an initial entry because an image can have a maximum of 256 different pixels from 0 to 255. It works better in the case of text compression reported in [84]. However, Saravanan et al. propose an image coding procedure utilizing LZW, which compresses an image in two stages shown in [41, 161, 162]. Firstly, a picture is encoded using Huffman coding. Secondly, after concatenating all the code words, LZW is applied to compress the encoded image, which provides a better result. However, the main challenge of that technique is to manage the string table.

This study uses a common numeric data set and shows the step-by-step details of implementation procedures of the entropy coding techniques, demonstrates comparisons among the methods, and explains the difficulties of the methods based on the experimental results. The organization of this study is shown as follows: the encoding and decoding procedure; and the analysis of run-length, Shannon–Fano, Huffman, LZW, and Arithmetic coding are discussed in Sections 5.2. The experimental results of some benchmarked images are explained in Section 5.2.6, and concluding statements are presented in Section 5.3.

5.2 Entropy Coding Techniques

5.2.1 Run-Length Coding

Run-length coding is a lossless compression procedure that takes the occurrence of data instead of statistical information, and it is generally utilized in TIFF and PDF formats reported in [140]. In the encoding, a single value and the count of the same consecutive values are stored. For instance, the encoding and decoding procedures are shown in Algorithm 1 and Algorithm 2, respectively, based on the 50 elements (A = [6 7 6 6 6 7 7 7 7 7 7 7 7 7 5 4 4 4 4 7 7 7 7 7 7 7 7 7 5 5 5 7 7 3 3 3 2 2 2 5 5 5 5 5 5 5 5 5 1 1]).

It shows that only twenty-six elements are preserved in two matrices instead of 50 items that designates that (26*8)=208 bits are sent to the decoder (50*8)= 400 bits. So, the average code length is 208/50 = 4.16 bits, and ((8-4.16)/8)*100 = 48% working memory is saved for the data set.

For example, the first 6 and 7 of the array (items) are reiterated once at positions 1 and 2, respectively. And the next 6 and 7 are repeated three times from positions 3 to 5 and 9 times from positions 6 to 14, respectively. This process will continue until the reading of all elements from the items array is finished. Conclusively, we get the same list as the original list(A) after decoding.

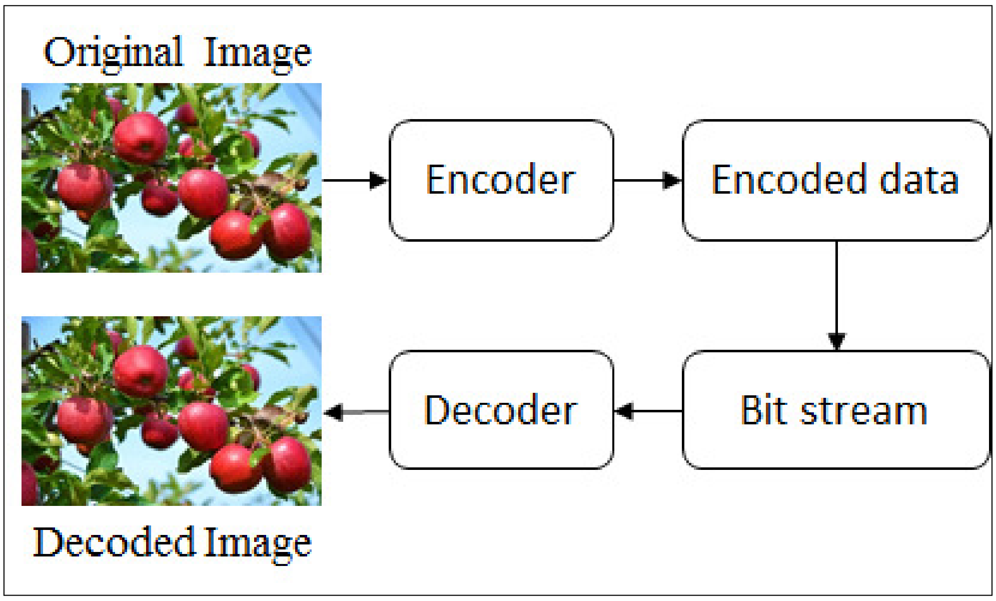


Figure 5.1: General block diagram of an image compression procedure.

```
Algorithm 1: Run-length encoding procedure
1 Calculate the difference (B = [1 -1 0 0 1 0 0 0 0 0 0 0 0 -2 -1 0 0 0 3 0 0 0 0 0 0 0 0 -2 0
  0 2 0 -4 0 0 -1 0 0 3 0 0 0 0 0 0 0 0 -4 0 1]) using $f(x) = f(x+1) - f(x)$;
2 Assign 1 to each non-zero data of B and we get B = [1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1
  0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1];
3 Save the positions of all ones into an array (position = [1 2 5 14 15 19 28 31 33 36 39
  48 50]) and the corresponding data into (items =[6 7 6 7 5 4 7 5 7 3 2 5 1]) from A.
  The arrays position and items are stored or sent as the encoded list of the original 50
  elements;
```

```
Algorithm 2: Run-Length Decoding Procedure
1 The two arrays position and items are received for decoding, and the decoder follows
  the style shown below for decompression;
2 Read each element from the array items and write the element repeatedly until its
  corresponding number in the position array is found;
```

Table 5.1: The results of Shannon-Fano encoding procedure

i	P_i	E_i	B_i	$\sum_{i=0}^{N-1} P_i\ B_i$	$P_i\ log_2\ P_i$	CR	BPP
7	0.42	00	2	0.84	-0.526	3.226	0.31
4	0.08	01	2	0.52	-0.292		
5	0.26	10	2	0.16	-0.505		
6	0.08	1100	4	0.32	-0.292		
3	0.06	1110	4	0.24	-0.244		
2	0.06	1111	4	0.24	-0.244		
1	0.04	1101	4	0.16	-0.186		
				L_{avg}=2.48	Entropy= 2.289		

Analysis of Run-length Coding Procedure

Run-length coding works well when an image contains long runs of identical samples that customarily does not appear in an authentic image which is the main quandary of run-length coding reported in [47, 141]. For example, the data(A) is rearranged with a slight change and the rearranged list is C = [1 6 7 6 6 7 7 7 4 7 7 7 7 5 7 4 4 7 4 7 7 7 7 7 7 7 7 5 7 7 5 5 7 7 3 3 2 3 2 2 5 5 5 5 6 5 5 5 5 5 1]. We apply run-length coding on C; and get [1 2 3 5 8 9 13 14 15 17 18 19 26 27 29 31 33 35 36 37 39 43 44 49 50] and [1 6 7 6 7 4 7 5 7 4 7 4 7 5 7 5 7 3 2 3 2 5 6 5 1] in position and items arrays . There is no compression here because the two arrays contain 50 elements together which is precisely identically tantamount to the initial list(C).

5.2.2 Shannon-Fano Coding

Shannon-Fano is a lossless coding technique that takes sorted probabilities in descending order of an image and separated them into two groups where each group's total sum is almost equivalent reported in [70].

Run-length coding does not perform any compression on the array C and the array contains seven different components (7, 5, 4, 6, 3, 2, 1) and their probabilities are 0.42, 0.26, 0.08, 0.08, 0.06, 0.06 and 0.04, respectively. As indicated by the algorithm, the two groups left (0.42, 0.08) and right (0.26, 0.08, 0.06, 0.06, 0.04) are made, and the Shannon-Fano encoding procedure is applied to the groups, demonstrated in Figure 5.2.

Efficiency is determined using the following equation 5.1, which is also utilized to measure a compression algorithm's performance. The array (C) encoded results appear in Table 5.1, where E_i represents the encoded code-word of i^{th} symbol.

$$efficiency = \frac{entropy}{L_{avg}} * 100\% \tag{5.1}$$

Using Table 5.1 The Shannon-Fano coding provides [1101110000110011000000000010000 0000100001010 0010000000000000001000001010000011101110111111101111111110101010 1100101010101 01101] bitstream of the data set (C), which is sent for decompression together with symbols and their probabilities. It looks that Shannon-Fano saves ((8-2.48)/8)*100 = 69% storage where run-length coding can save no memory for the same data set. So, Shannon-Fano provides a 69% better result than run-length coding for the data set, and the algorithm proficiency is 92.298%.

In decoding, Shannon-Fano receives the encoded bitstream, items, and their relating probabilities. It builds a similar tree as Figure 5.2 dependent on the probabilities. Finally, we get the same data list as array C.

Table 5.2: Huffman encoding procedure

i	P_i	E_i	B_i	$\sum_{i=0}^{N-1} P_i\ B_i$	$P_i\ log_2 P_i$	CR	BPP
7	0.42	1	1	0.42	-0.526		
5	0.26	01	2	0.52	-0.505		
4	0.08	0001	4	0.32	-0.292		
6	0.08	0010	4	0.32	-0.292	3.448	0.29
3	0.06	0011	4	0.24	-0.244		
2	0.06	00000	5	0.3	-0.244		
1	0.04	00001	5	0.2	-0.186		
				L_{avg}=2.32	Entropy= 2.289		

Analysis of Shannon-Fano Coding

In Shannon-Fano coding, we cannot be sure about the codes generated. There might be two different codes for a same symbol depending on the way we build our tree shown below through an example. Assume two groups (7=.42 and **6=.08**) and (5=.26, **4=.08**, 2=.06, 3=.06, 1=.04) are made instead of (7=.42 and **4=.08**) and (5=.26, **6=.08**, 2=.06, 3=.06, 1=.04) simply exchanging two probabilities between two groups appeared in bold . And if we apply Shannon-Fano decoding on the received bitstream, we get (D = [1 **4** 7 **4 4** 7 7 7 **6** 7 7 7 7 7 5 7 **6 6** 7 **6** 7 7 7 7 7 7 7 5 7 7 5 5 7 7 **2 2 3 2 3 3** 5 5 5 5 **4** 5 5 5 5 5 1]) as decoded values. In the decoded list, the bold symbols represent the changed components of the original list which are considered as loss. There are 14 elements in the decoded list that the Shannon-Fano's rebuilt tree can not reproduce perfectly. So it losses (14/50)*100= 28% data for only 50 elements.

5.2.3 Huffman coding

Shannon-Fano coding sometimes produces the most flawed code for some probabilities set because it cannot produce an optimal tree. David A. Huffman illustrated a coding procedure that consistently makes an optimal tree and tackles the issues in Shannon-Fano coding reported in [19, 163]. Shannon-Fano coding is a top-down methodology, whereas Huffman coding uses the reverse route, from the leaves to the root. Huffman coding uses the statistical information of an image like Shannon-Fano coding. Figure 5.3 and Table 5.2 demonstrate a graphical representation of the Huffman tree, and the outcomes depend on the same data used in Shannon-Fano coding.

Based on Table 5.2, Huffman produces [000010010101001000101 110001111101100010001100 0011111111011101011100110011 10000000110000000000001010101001001010101010100001] as encoded bitstream for the data set (C) that is sent for decoding with symbols and their corresponding probabilities. In this way, Huffman coding saves 71% memory space, which is 69% and 2% more than Run-length and Shannon-Fano coding, respectively and the efficiency of Huffman coding is 98.664% which is 6.366% more than Shannon-Fano coding.

Huffman coding provides an optimal prefix code and receives encoded bitstream, items, and corresponding probabilities and uses the following methodology for decompression. Finally, Huffman produces the indistinguishable data as the original list (C).

Analysis of Huffman Coding

The main problem of Huffman coding is that it is very sensitive to noise. A minor change in any bit of the encoded bitstream would break the whole message reported in [**?**]. Assume the decoder receives items, probabilities, and the encoded bitstream with only three altered bits at the positions 5^{th}, 19^{th}, 54^{th}. Then we get [**2** 6 7 6 6 **5** 7 **4 7** 7 7 7 **5 7 4** 4 **7 4 7** 7 7 7 7 7 7 7 **3** 5 **5** 7 **7 3 3 2** 3 **2** 2 **5 5 5** 5 **6** 5 5 **5** 5 5 **1**] as decoded values where bold elements (total 23) indicate

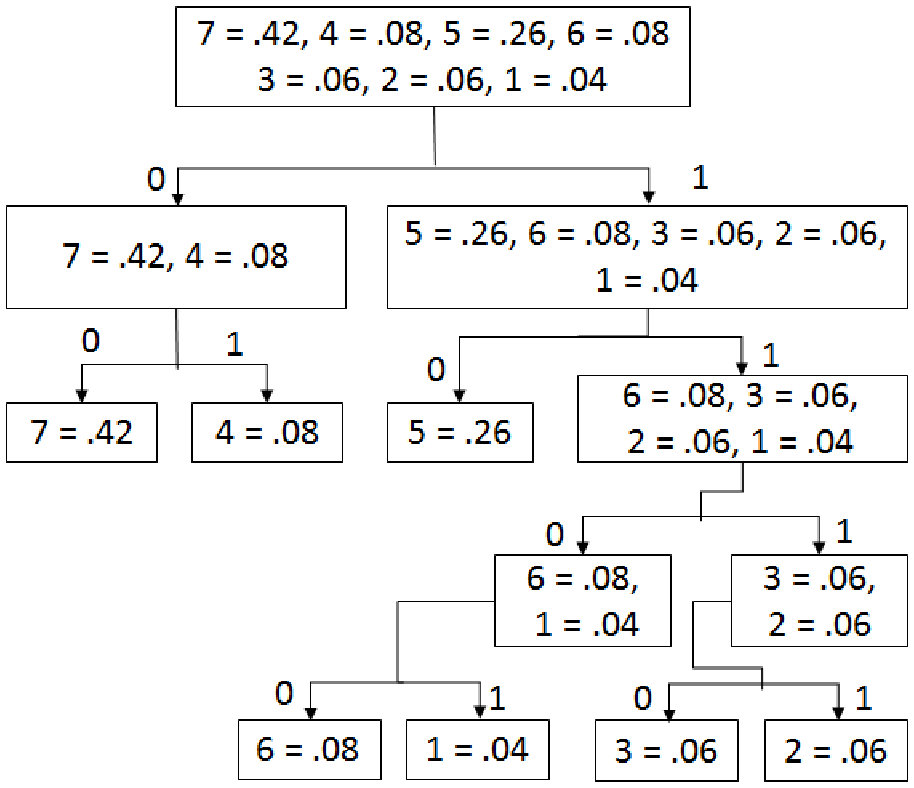


Figure 5.2: Encoding procedure of Shannon-Fano

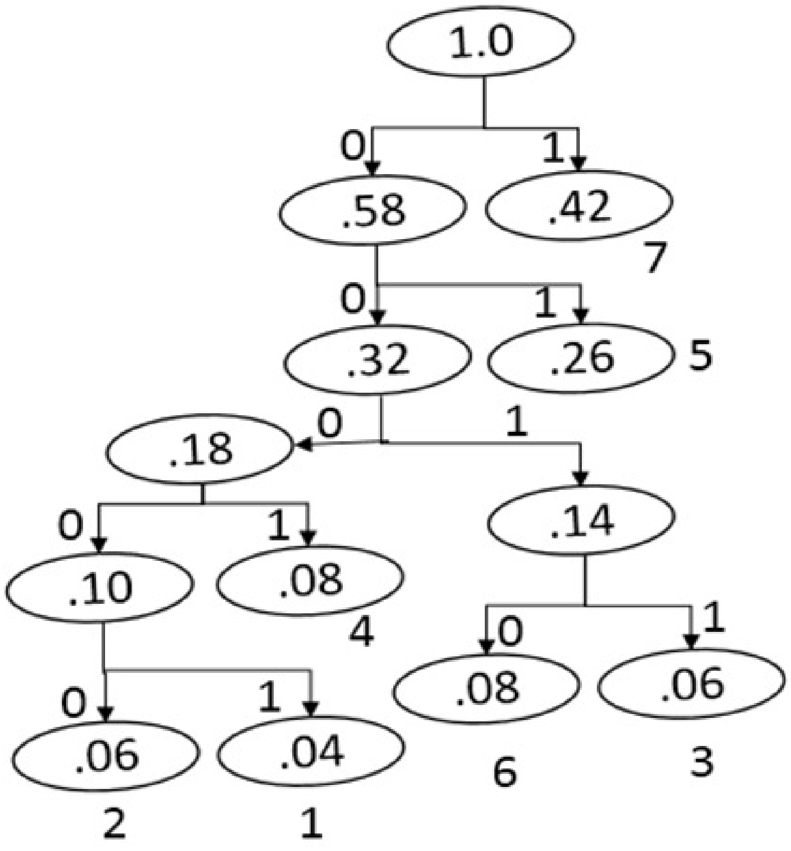


Figure 5.3: Huffman tree for encoding

loss of data. In addition, it produces only 47 elements rather than 50 elements. So it devastates ((23+3)/50)*100 = 52% data.

5.2.4 Lempel–Ziv–Welch (LZW) Coding

Lempel–Ziv–Welch (LZW) is generally used for lossless text compression technique used to encodes a sequence of characters with a unique code using a table-based lookup algorithm. This algorithm works better in text compression and provides most noticeably a terrible outcome for another sort of compression. Since the previously mentioned original list (C) contains only 7 (1–7) different values, only 1–7 are inserted into the table as an initial dictionary first. Applying the LZW encoding procedure on C shown in Table 5.3 and we get the decoded list that appears in Table 5.4. Finally, the encoded bitstream is sent to the decoder. Each piece of encoded data is converted into a 6-bit binary because the biggest value is 33 in the encoded list, and just 6 bits are required to represent 33.

Since the average code length is 3.84, as shown in Table 5.4, LZW saves 36% memory, which is 28.7356% and 29.3103% more than Shannon–Fano and Huffman coding individually for the same dataset. Furthermore, the only encoded bitstream is sent to the decoder for decompression.

For instance, the mentioned encoded bitstream converts every six bits into decimal value and is assigned 1–7 as the initial dictionary is shown in Table 5.5. The decoding demonstration for the encoded data (Code) is shown in Table 5.6, and we get a similar list as C after decoding.

Analysis of LZW Coding

The searching dictionary is a significant challenge in the LZW compression technique because it is more complicated and time-consuming. Moreover, an image that does not carry many repetitive data at all cannot be reduced, and it is suitable for deducing file size that carries more repeated data reported in [73, 74].

5.2.5 Arithmetic Coding

Arithmetic coding is a lossless data compression procedure where a set of symbols is presented using a fixed number of bits reported in [75, 78]. The original array (C) contains 50 elements. The encoding style of 50 items in a figure is challenging, which is why only the encoding style for ten items is shown. Suppose that the 10-item list is [2 3 4 3 4 4 4 1 4 1]. There are four different items (4 3 1 2) on the list, and their corresponding probabilities are 0.5, 0.2, 0.2, 0.1, individually. The four elements (4 3 1 2) contain 50%, 20%, 20% and 10% data, respectively. Thus, each limit is divided into 50%, 20%, 20%, and 10% each time to encode each element, shown in Figure 5.4 for all ten elements.

For the example shown in Figure 5.4, the LLL and LUL are 0.9551925 and 0.9551975. Thus, the tag is 0.955195. The bitstream of the tag value is 001111000110111. Thus, the average code length is 15/10 = 1.5 bits, and the compression ratio is 5.3333. Finally, the tag's bitstream, symbols (4,3,1,2), and their corresponding probabilities (0.5, 0.2, 0.2, 0.1) are sent to the decoder for decompression. When Arithmetic coding is applied on data set (C), it produces [000001011001101011101001011111001010101110111110011 111101010110010111001100 1 11000000001010110100101 10 110111100001011] bitstream from the provided tag. Thus, the average code length and compression ratios are 2.3000 bits and 3.4783 separately, which saves 71.25% of storage. It appears that run-length, Shannon–Fano, Huffman, and LZW coding use 44.7115%, 7.2581%, 6.5041%, and 33.908% more memory than arithmetic coding.

The decoding procedure of arithmetic coding receives tag, symbols, and corresponding probabilities, and the tag is converted into its floating-point number. For decompression, if the tag is in between in any range, then the range's symbol is taken as the decoded value. The

Table 5.3: LZW encoding procedure.

Row Number	**Encoded Output**	**Dictionary**	
		Index	**Entry**
1	-	**1**	**1**
2	-	**2**	**2**
3	-	**3**	**3**
4	-	**4**	**4**
5	-	**5**	**5**
6	-	**6**	**6**
7	-	**7**	**7**
8	1	8	16
9	6	9	67
10	7	10	76
11	6	11	66
12	9	12	677
13	7	13	77
14	7	14	74
15	4	15	47
16	13	16	777
17	13	17	775
18	5	18	57
19	14	19	744
20	15	20	474
21	15	21	477
22	16	22	7777
23	16	23	7775
24	18	24	577
25	7	25	75
26	5	26	55
27	24	27	5773
28	3	28	33
29	3	29	32
30	2	30	23
31	29	31	322
32	2	32	25
33	26	33	555
34	26	34	556
35	6	35	65
36	33	36	5555
37	26	37	551
38	1	-	-
39	0	Stop Code	

Table 5.4: Average code length and compression ratio.

Encoded Data	Encoded Bit's Stream (6 Bits Each)	ACL	CR
	0000010001100001110001100010		
1 6 7 6 9 7 7 4 13	0100011100011100010000110100		
13 5 14 15 15 16	1101000101001110001111001111		
16 18 7 5 24 3 3 2	01000000100000100100001110001	3.84	2.083
29 2 26 26 6 33 26	0101100000001100001100001001		
1 0	1101000010011010011010000110		
	1000010111010000001000000		

Table 5.5: Initial dictionary.

Initial Dictionary	
Index	Entry
1	1
2	2
3	3
4	4
5	5
6	6
7	7

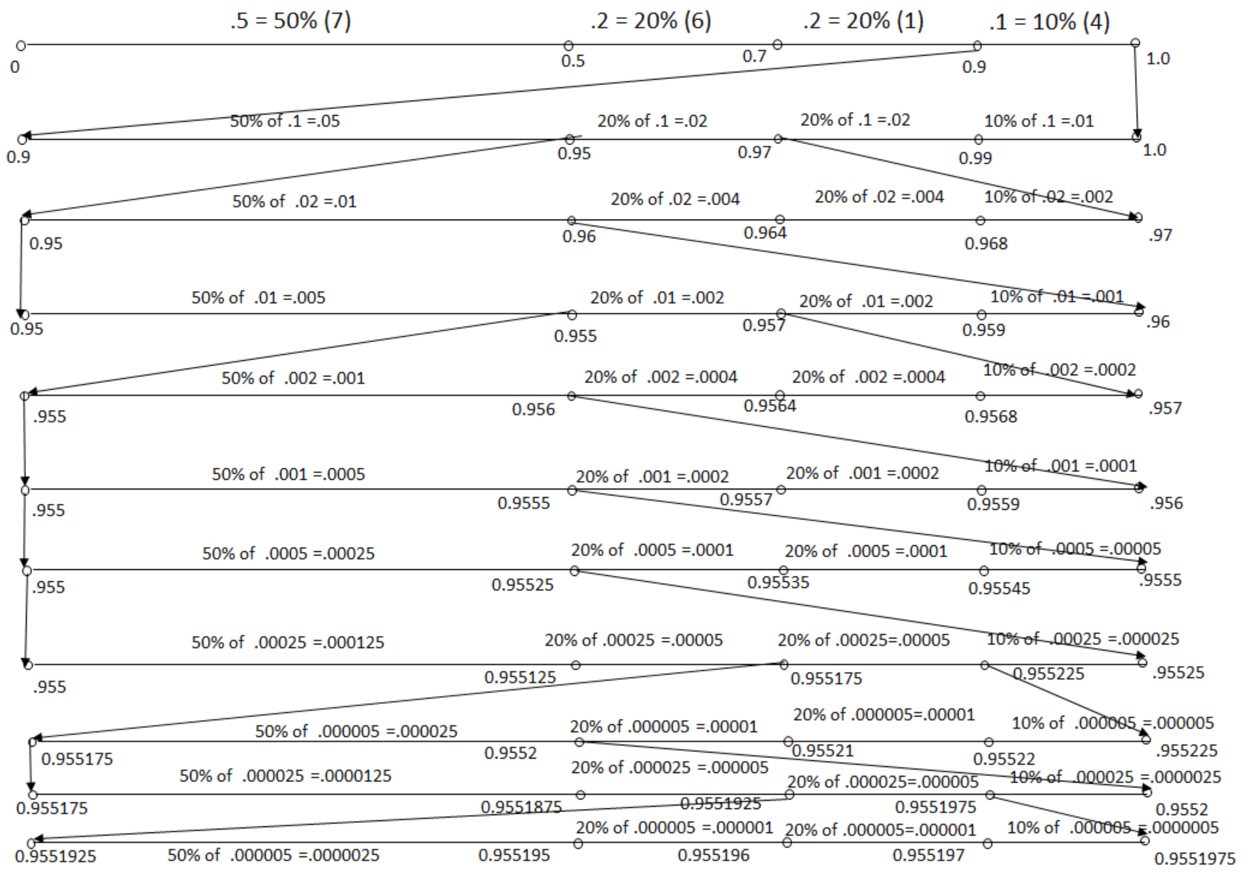


Figure 5.4: Arithmetic encoding procedure.

Table 5.6: The decoding procedure of LZW coding

Row Number	Code	Output	Full	Conjecture
1	1	1		8: 1?
2	6	6	8: 16	9: 6?
3	7	7	9: 67	10: 7?
4	6	6	10: 76	11: 6?
5	9	67	11:66	12: 67?
6	7	7	12: 677	13: 7?
7	7	7	13:77	14: 7?
8	4	4	14:74	15: 4?
9	13	77	15:47	16: 77?
10	13	77	16: 777	17: 77?
11	5	5	17:775	18:5?
12	14	74	18:57	19:74?
13	15	47	19:744	20:47?
14	15	47	20:474	21:47?
15	16	777	21:477	22:777?
16	16	777	22:7777	23:777?
17	18	57	23:7775	24: 57?
18	7	7	24:577	25:7?
19	5	5	25:75	26: 5?
20	24	577	26:55	27:577?
21	3	3	27:5773	28:3?
22	3	3	28:33	29:3?
23	2	2	29: 32	30: 2?
24	29	32	30:23	31: 32?
25	2	2	31:322	32:2?
26	26	55	32:25	33:55?
27	26	55	33:555	34:55?
28	6	6	34:556	35:6?
29	33	555	35:65	36:555?
30	26	55	37:5555	38:55?
31	1	1		

whole decoding procedure of the ten values is demonstrated in the following list using Figure 5.5, and we get the exact list [2 3 4 3 4 4 4 1 4 1] like the original. Here, the floating value of the corresponding tag's bitstream is 0.955195.

1. tag = 0.955195. Since .9 < = tag < = 1.0, Thus, decoded value is 2 because the symbol 2 is in range.
2. NT1 = (tag-LL)/r = 0.55195 and it is in between 0.5 and 0.7, so the decoded value is 3.
3. NT2 = (NT1 - LL)/r = 0.25975 and it is in between 0 and 0.5, so the decoded value is 4.
4. NT3 = (NT2 - LL)/r = 0.5195 and it is in between 0.5 and 0.7, so the decoded value is 3.
5. NT4 = (NT3 - LL)/r = 0.0975 and it is in between 0 and 0.5, so the decoded value is 4.
6. NT5 = (NT4 - LL)/r = 0.195 and it is in between 0 and 0.5, so the decoded value is 4.
7. NT6 = (NT5 - LL)/r = 0.39 and it is in between 0 and 0.5, so the decoded value is 4.
8. NT7 = (NT6 - LL)/r = 0.78 and it is in between 0.7 and 0.9, so the decoded value is 1.
9. NT8 = (NT7 - LL)/r = 0.4 and it is in between 0 and 0.5, so the decoded value is 4.
10. NT9 = (NT8 - LL)/r = 0.8 and it is in between 0.7 and 0.9, so the decoded value is 1.

Analysis of Arithmetic Coding Procedure

The authors in [77, 78] state that arithmetic coding provides a better compression ratio. However, it takes so much time that it is virtually not utilizable for dynamic compression. Furthermore, its use is restricted by the patent. On the other hand, though Huffman coding provides marginally less compression, it utilizes much less time to encode an image than arithmetic coding. That is why it is good for dynamic compression reported in [78, 79]. Furthermore, an image encoded by arithmetic coding can corrupt the entire image for a single bit error because it has a very impecunious error resistance reported in [80–82]. Another problem is that an entire code-word must be taken to start interpreting a message. Contiguous to the direct inhibition of entropy coding is that it increments CPU's complexity stated in [83, 84]. Suppose the decoder receives the tag of the original 50 elements with only a first bit altered, and we get [**6 5 4 2 7** 7 7 7 **7 2** 7 **5 6 7** 7 **7 7 4 7** 7 7 7 7 7 **1** 7 **7 6 5 7** 5 7 7 **7 2 4 7** 2 **7** 5 **7** 5 **7 1** 5 **7 1 7 1** 1] as a decoded list where the bold symbols indicate the altered values. In the list, 31 elements have been altered, which means (31/50) $\times$ 100 = 62% of the data have been corrupted.

5.2.6 Experimental Results and Analysis

The outcomes and investigation of entropy coding methods have been demonstrated in this segment. The entropy coding techniques have been applied to the different types of benchmarked images. In this study, we have initially used three PC-created photographs. The next twenty-two medical images from the DICOM Image dataset [164] of various sizes appeared in Figure 5.6. Encoding time, decoding time, average code length, compression ratio, PSNR, and efficiency have been used to analyze the algorithms' performance.

The encoding and decoding time are the periods of time required to encode and decode an image. Average code length determines the number of bits used to store a pixel on average, and the compression ratio represents the ratio of original and compressed images. Pick signal-to-noise ratio ((PSNR)) is used to measure the quality of an image. Less encoding and decoding time, short average code length, and higher compression ratio tell how much faster an algorithm

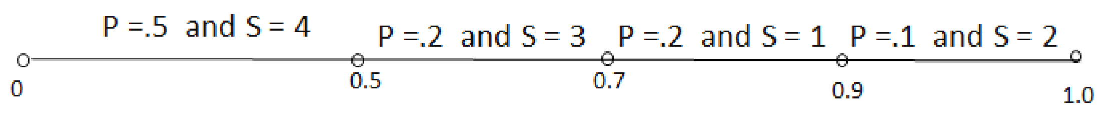


Figure 5.5: Arithmetic decoding procedure

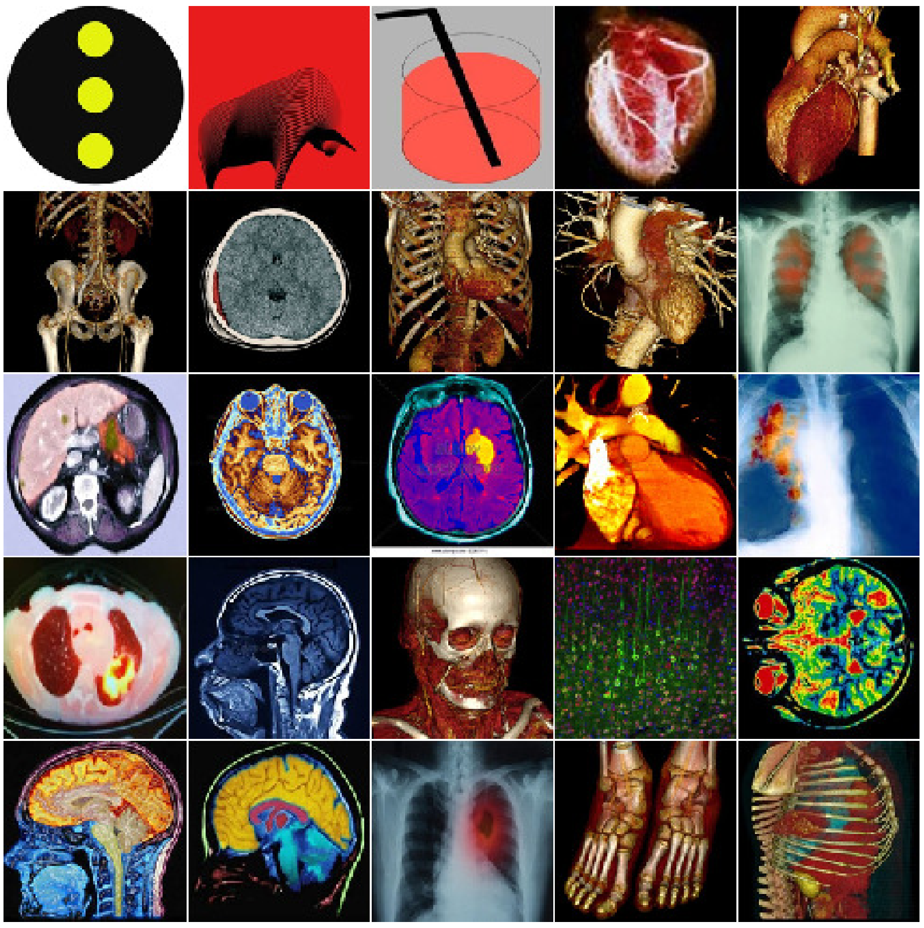


Figure 5.6: Original image list.

is and how much less memory it uses. The higher efficiency and PSNR convey that an image contains high-quality information. The encoding time, decoding time, average code length, and compression ratio are shown in Tables 5.7–5.10. Figures 5.7–5.10 show the graphical representation of encoding time, decoding time, average code length, compression ratio, and efficiency, respectively, based on the twenty-five images.

Table 5.7 shows that arithmetic and run-length coding take the highest (4.0178) and lowest (0.1349) milliseconds on average. In contrast, Shannon–Fano, Huffman, and LZW take 0.5873, 0.2488, and 0.1054 milliseconds, respectively, to encode the images. It appears that arithmetic coding uses 96.6424%, 85.3825%, 93.8076% and 97.3767% more time than run-length, Shannon–Fano, Huffman, and LZW coding, respectively. However, Huffman coding uses much less time (0.0062) on average in decoding, whereas arithmetic coding uses more time, which is demonstrated in Table 5.8. On the other hand, LZW uses more time than Shannon–Fano and Huffman coding but less than Arithmetic and Run-Length coding. Figures 5.7 and 5.8 show the graphical representation of encoding and decoding time for comparison.

Tables 5.9 and 5.10 show average code length and compression ratio, respectively. It looks that RLE uses 10.5618 bits per pixel, on average, which is 24.2553% more memory being used than the original images, which is the reason it is not used directly for real image compression. On the other hand, LZW uses the lowest number of bits (5.9365) per pixel, but the problem of LZW is that it sometimes uses more memory than an original, which happened for image 21 shown in Table 5.9. Arithmetic coding uses the second-lowest number of bits per pixel on average. Thus, arithmetic coding is the best coding technique because it provides a better compression ratio than other state-of-the-art techniques without LZW shown in Table 5.10. Figures 5.9 and 5.10 demonstrate the graphical representation of average code length and compression ratio separately for comparison.

All the entropy coding strategies are lossless. Thus, the pick signal-to-noise ratio and mean squared error (MSE) for each algorithm are inf and zero, respectively, for every case. However, arithmetic and run-length coding, on average, have the highest (99.9899) and lowest (58.6783) efficiency than the other methods shown in Figure 5.11. Even though LZW coding's proficiency provides better outcomes and sometimes provides terrible results, it is not used for image compression in real applications. The list of the decompression images is shown in Figure 5.12.

From the previously mentioned perspectives, arithmetic coding is the best way when more compression is required; however, it isn't helpful for a real-time application because of taking additional time in encoding and decoding steps. Searching in a dictionary is a big challenging issue for LZW coding, and it provides the worst results for image compression. Shannon–Fano coding sometimes does not give optimal code and provides two different codes for the same element, which is why it is obsolete now. Run-length coding is not suitable for a straightforward real image compression.

Thus, it very well may be reasoned that Huffman coding is the best algorithm for the recent technologies among the state-of-the-art lossless methods mentioned used in various applications. However, if we can decrease the encoding and decoding time in arithmetic coding, it will be the best algorithm. On the other hand, Huffman coding will work more if we can decrease its average code length keeping its same encoding and decoding times.

In this research work, all the experiments are done using C, Matlab (version 9.4.0.813654 (R2018a). Natick, Massachusetts, USA: The MathWorks Inc.; 2018) and Python languages. For the coding environments, Spyder (Python 3.6), Codeblocks (17.12, The Code::Blocks Team) and Matlab are utilized. Furthermore, we utilized an HP laptop (Palo Alto, California, United States) that contained the Intel Core i3-3110M @2.40 GHz processor (Santa Clara, USA), 8 GB DDR3 RAM, 32 KB L1D-Cache, 32 KB L1I-Cache, 256 KB L2 Cache and 3 MB L3 Cache, where L1D, L1I, and L2 Caches contained 8-way set associative, 64-byte line size each, and L3 Cache contained 12-way set associative, 64-byte line size. According to the algorithms used for testing, the CPU-Time is $1.499 \times 10^{-6} O(P)$, $6.481 \times 10^{-6} O(P + |\beta| * log|\beta|)$,

Table 5.7: Encoding time comparison.

Images	RLE	Shannon–Fano	Huffman	LZW	Arithmetic
1	0.171	0.8667	0.2056	0.123	5.5032
2	0.167	0.7524	0.1304	0.105	2.9515
3	0.121	0.6455	0.2673	0.109	2.223
4	0.027	0.2983	0.4699	0.022	0.3101
5	0.167	0.6735	0.215	0.106	3.7628
6	0.187	0.7304	0.2534	0.106	3.3215
7	0.141	0.6262	0.1925	0.105	2.9568
8	0.165	0.7816	0.2183	0.118	4.6419
9	0.186	0.6002	0.2252	0.107	4.4352
10	0.137	0.5079	0.1816	0.106	7.3937
11	0.126	0.4753	0.2182	0.106	4.5515
12	0.096	0.449	0.2545	0.106	2.9656
13	0.113	0.4942	0.2034	0.11	5.2077
14	0.161	0.8058	1.0607	0.108	5.525
15	0.102	0.5208	0.1932	0.106	4.3877
16	0.112	0.4978	0.1979	0.106	3.8302
17	0.092	0.4684	0.1939	0.106	4.5352
18	0.186	0.6756	0.2139	0.118	5.9698
19	0.189	0.687	0.166	0.116	5.7538
20	0.086	0.4395	0.2088	0.111	2.227
21	0.112	0.5085	0.2059	0.106	3.217
22	0.103	0.4413	0.2007	0.11	2.5256
23	0.122	0.5298	0.1617	0.105	3.8022
24	0.172	0.6697	0.2004	0.107	4.7927
25	0.132	0.5369	0.1818	0.106	3.6537
Average	**0.1349**	**0.5873**	**0.2488**	**0.1054**	**4.0178**

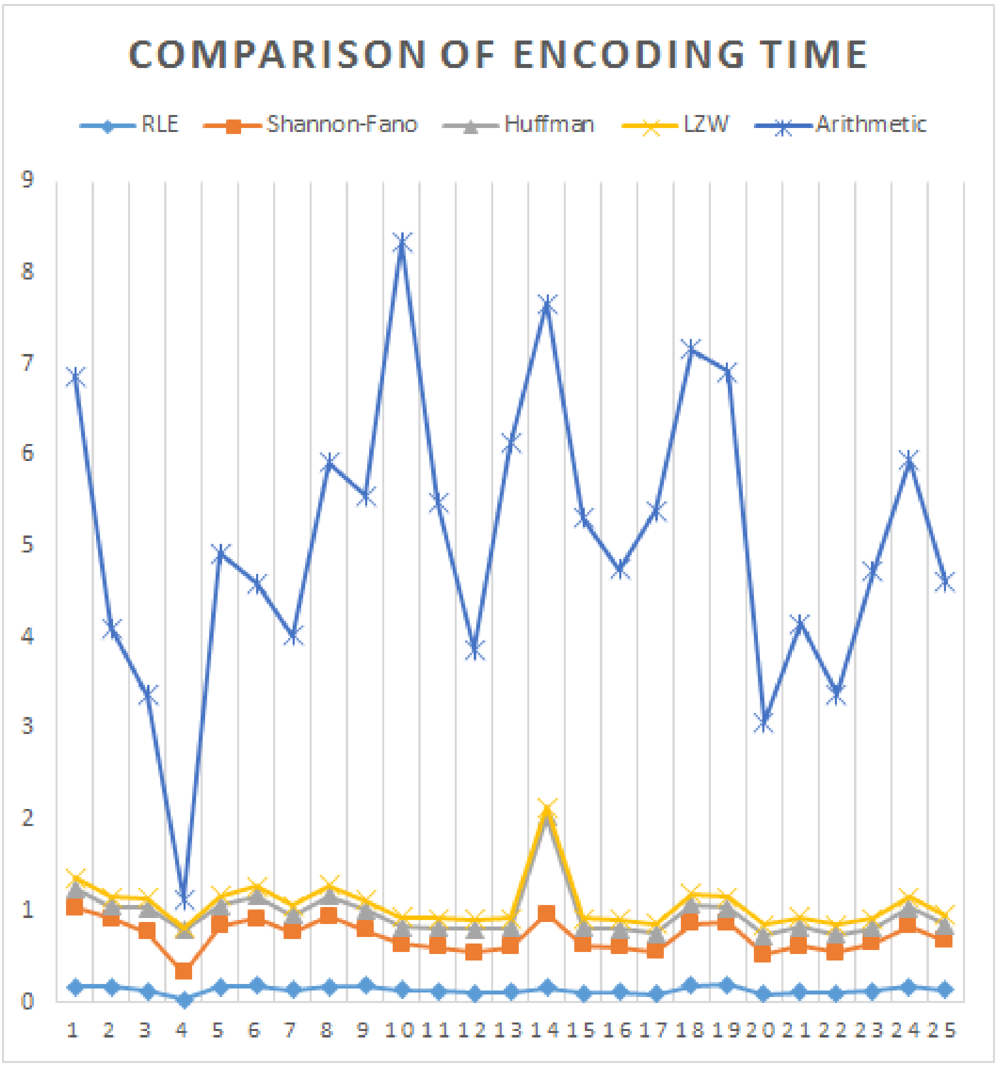


Figure 5.7: Encoding time comparison of the images.

Table 5.8: Decoding time comparison.

Images	RLE	Shannon–Fano	Huffman	LZW	Arithmetic
1	0.059	0.0061	0.0029	0.009	6.2899
2	0.048	0.0056	0.0038	0.013	3.273
3	0.048	0.005	0.0047	0.012	2.774
4	0.01	0.0021	0.011	0.002	0.3718
5	0.058	0.0082	0.0072	0.106	4.5912
6	0.078	0.0077	0.007	0.024	4.1704
7	0.052	0.0075	0.0071	0.021	3.6072
8	0.066	0.0096	0.0084	0.037	5.7222
9	0.07	0.0059	0.0093	0.033	5.4118
10	0.059	0.0046	0.0051	0.029	5.6243
11	0.049	0.0065	0.0089	0.024	5.347
12	0.029	0.0055	0.0056	0.017	3.3815
13	0.036	0.0065	0.0049	0.019	4.7372
14	0.055	0.0094	0.0078	0.036	7.6486
15	0.038	0.0071	0.0034	0.024	5.0222
16	0.036	0.0072	0.0038	0.022	4.5165
17	0.038	0.0032	0.0056	0.019	4.7644
18	0.064	0.0044	0.0116	0.032	9.3636
19	0.071	0.0101	0.0043	0.041	6.7193
20	0.031	0.0064	0.0092	0.016	2.7133
21	0.038	0.0031	0.0032	0.024	4.2221
22	0.037	0.0056	0.0041	0.015	2.8519
23	0.05	0.0074	0.0037	0.026	4.696
24	0.059	0.0043	0.0074	0.033	5.7915
25	0.058	0.0079	0.004	0.03	4.4087
Average	**0.0495**	**0.0063**	**0.0062**	**0.0266**	**4.7208**

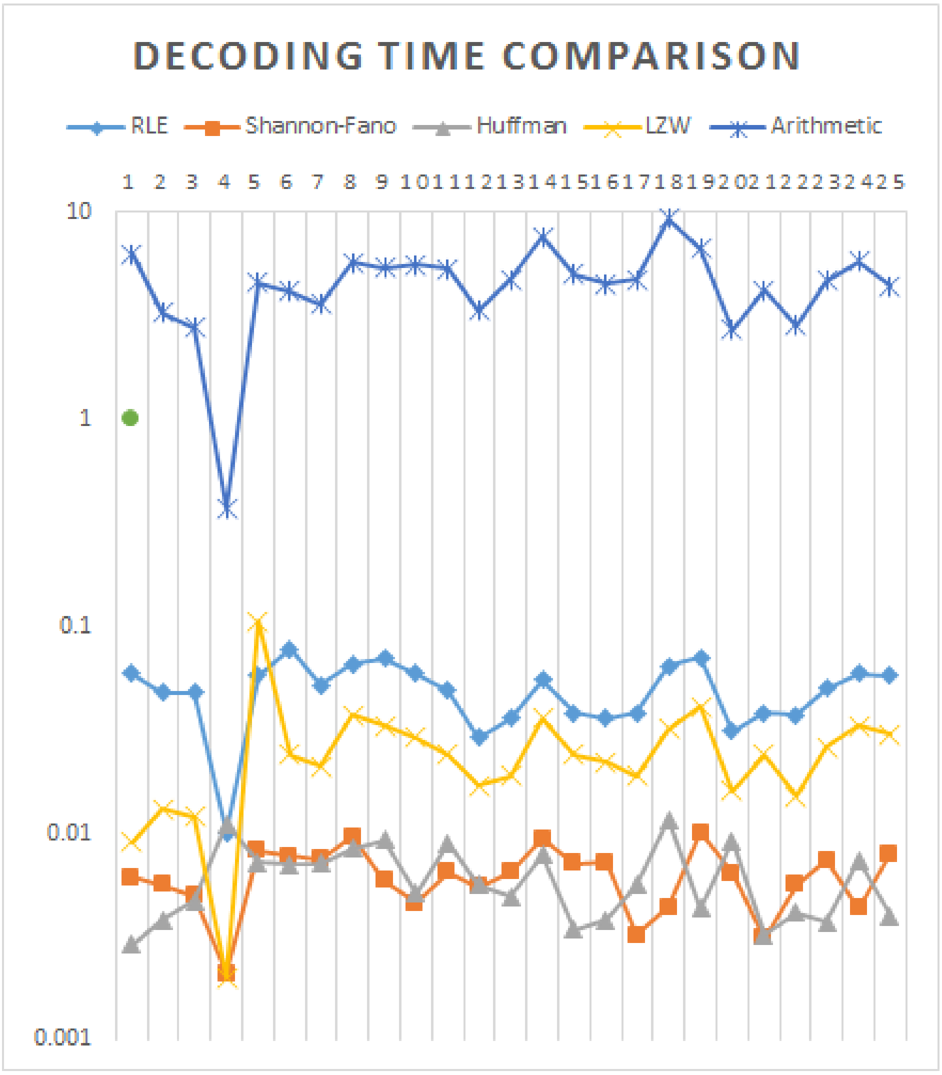


Figure 5.8: Decoding time comparison of the images.

Table 5.9: Comparison of average code length.

Images	RLE	Shannon–Fano	Huffman	LZW	Arithmetic
1	2.6114	2.861	2.4394	1.554	2.4265
2	5.0743	3.649	3.3302	2.8331	3.264
3	5.9338	4.035	3.6893	3.2044	3.6267
4	11.6135	6.652	6.2437	7.3533	6.2264
5	8.8868	5.904	5.349	5.0304	5.3195
6	7.9404	5.429	4.6825	4.3298	4.672
7	8.5559	5.614	4.9738	4.8468	4.9537
8	12.194	7.04	6.529	6.4582	6.4999
9	11.0768	6.557	6.1968	6.0463	6.1744
10	12.1297	7.857	7.4268	7.1652	7.3972
11	12.8617	7.733	7.2676	7.5491	7.2354
12	8.4888	5.887	5.3107	5.2507	5.2929
13	10.3832	6.661	6.1475	5.7646	6.1093
14	11.4108	7.272	6.7362	6.2315	6.6092
15	11.2102	7.936	7.4703	7.1055	7.4378
16	11.1044	7.825	7.3288	7.0915	7.3002
17	11.5137	7.056	6.6154	6.5353	6.5865
18	10.7582	6.724	6.3173	6.0833	6.2888
19	15.3026	6.781	6.2937	7.0633	6.2509
20	11.6004	6.951	6.3686	6.4392	6.3459
21	14.3268	7.831	7.3847	8.0181	7.3486
22	11.5411	6.635	6.1382	6.1512	6.1147
23	13.2045	7.845	7.3723	7.2199	7.3443
24	10.4551	6.503	6.0551	5.7253	6.0129
25	13.8657	7.612	7.1845	7.3613	7.1488
Average	**10.5618**	**6.514**	**6.0341**	**5.9365**	**5.9995**

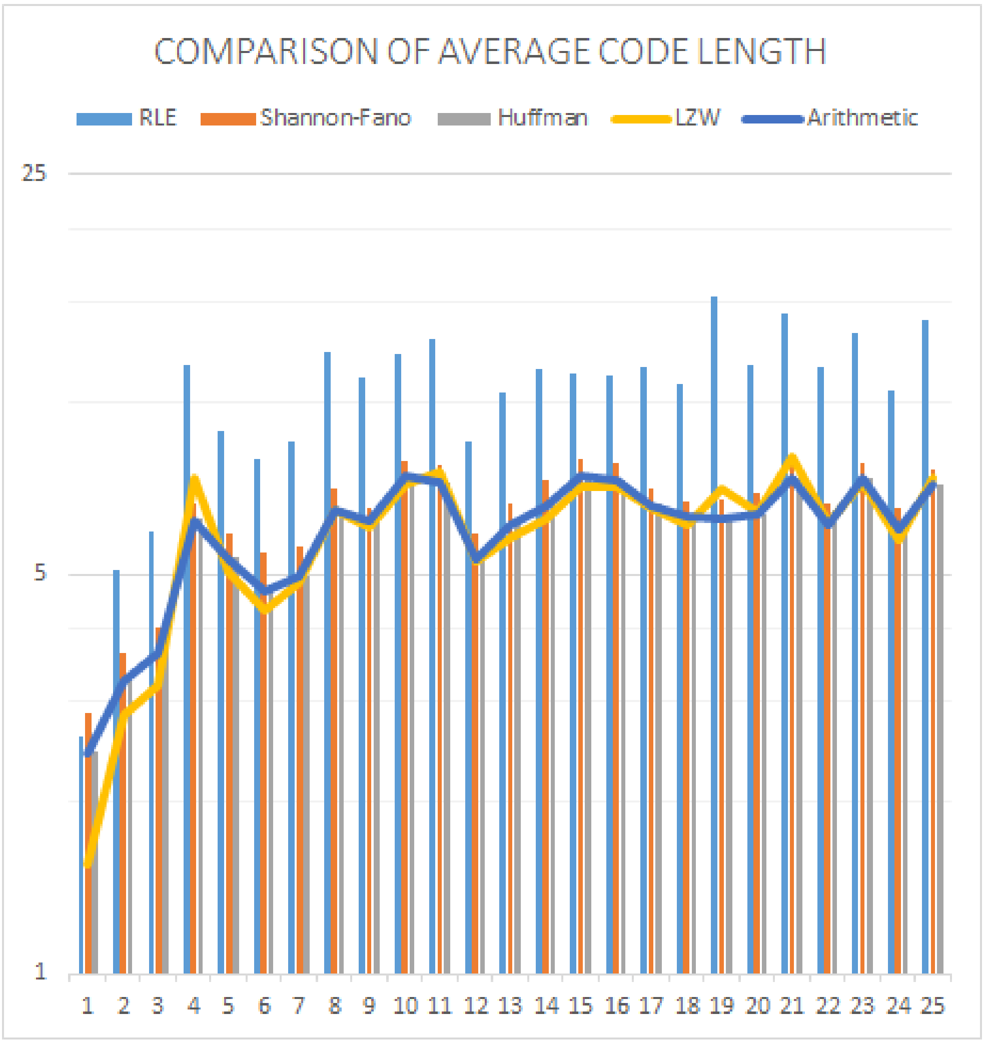


Figure 5.9: Average code length comparison of the images.

Table 5.10: Comparison of compression ratio.

Images	**RLE**	**Shannon–Fano**	**Huffman**	**LZW**	**Arithmetic**
1	3.0635	2.7961	3.2795	5.1481	3.2969
2	1.5766	2.1924	2.4023	2.8237	2.451
3	1.3482	1.9825	2.1684	2.4966	2.2059
4	0.6889	1.2026	1.2813	1.0879	1.2849
5	0.9002	1.3551	1.4956	1.5903	1.5039
6	1.0075	1.4737	1.7085	1.8477	1.7123
7	0.935	1.425	1.6084	1.6506	1.615
8	0.6561	1.1364	1.2253	1.2387	1.2308
9	0.7222	1.2201	1.291	1.3231	1.2957
10	0.6595	1.0181	1.0772	1.1165	1.0815
11	0.622	1.0346	1.1008	1.0597	1.1057
12	0.9424	1.3589	1.5064	1.5236	1.5115
13	0.7705	1.201	1.3014	1.3878	1.3095
14	0.7011	1.1002	1.1876	1.2838	1.2104
15	0.7136	1.0081	1.0709	1.1259	1.0756
16	0.7204	1.0223	1.0916	1.1281	1.0959
17	0.6948	1.1337	1.2093	1.2241	1.2146
18	0.7436	1.1898	1.2664	1.3151	1.2721
19	0.5228	1.1798	1.2711	1.1326	1.2798
20	0.6896	1.1509	1.2562	1.2424	1.2607
21	0.5584	1.0216	1.0833	0.9977	1.0886
22	0.6932	1.2058	1.3033	1.3006	1.3083
23	0.6059	1.0198	1.0851	1.1081	1.0893
24	0.7652	1.2301	1.3212	1.3973	1.3305
25	0.577	1.0509	1.1135	1.0868	1.1191
Average	**0.875128**	**1.30838**	**1.428224**	**1.545472**	1.43798

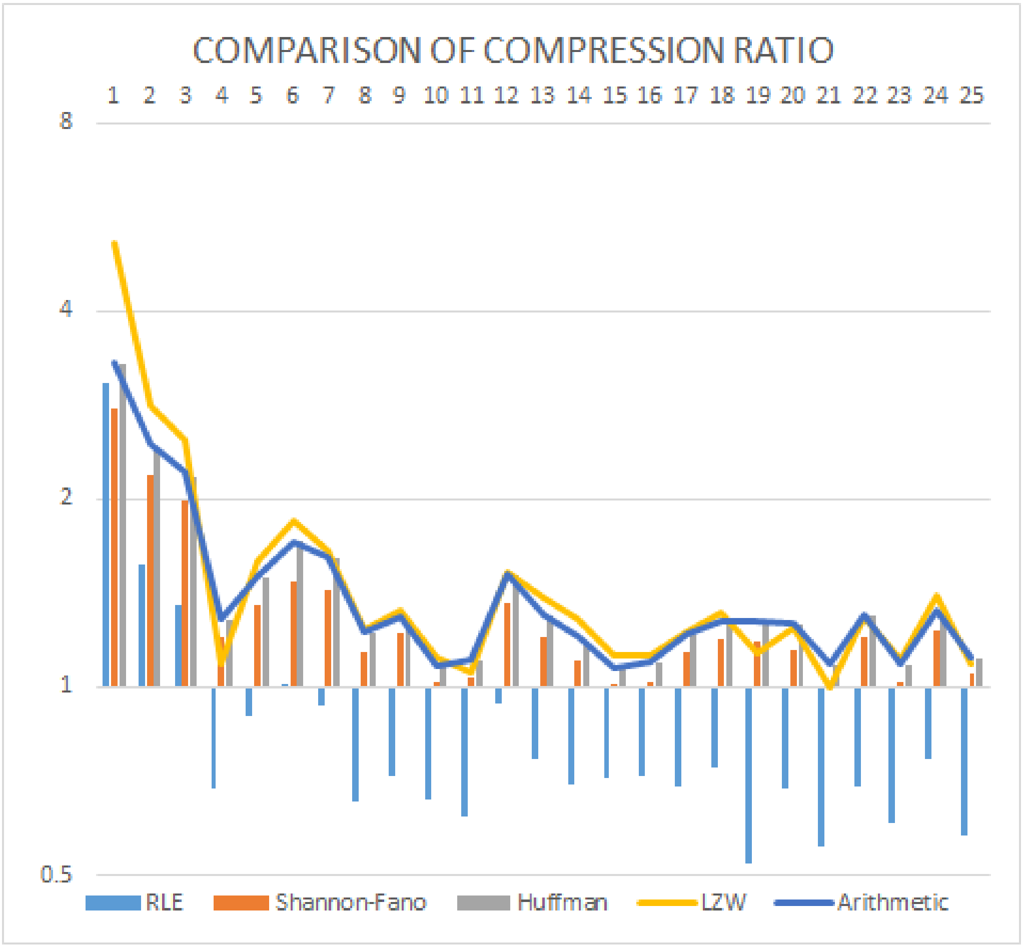


Figure 5.10: Comparison of compression ratio.

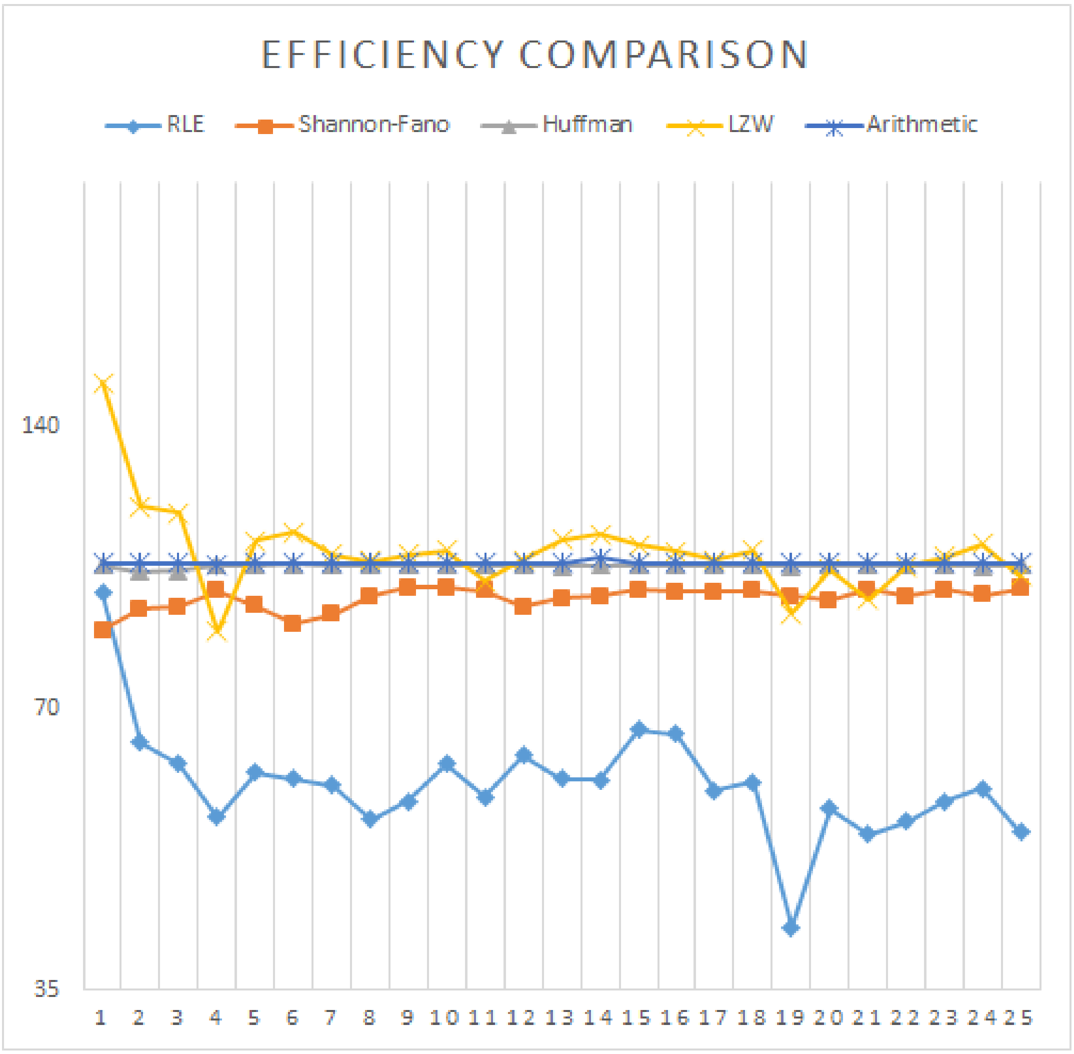


Figure 5.11: Efficiency comparison.

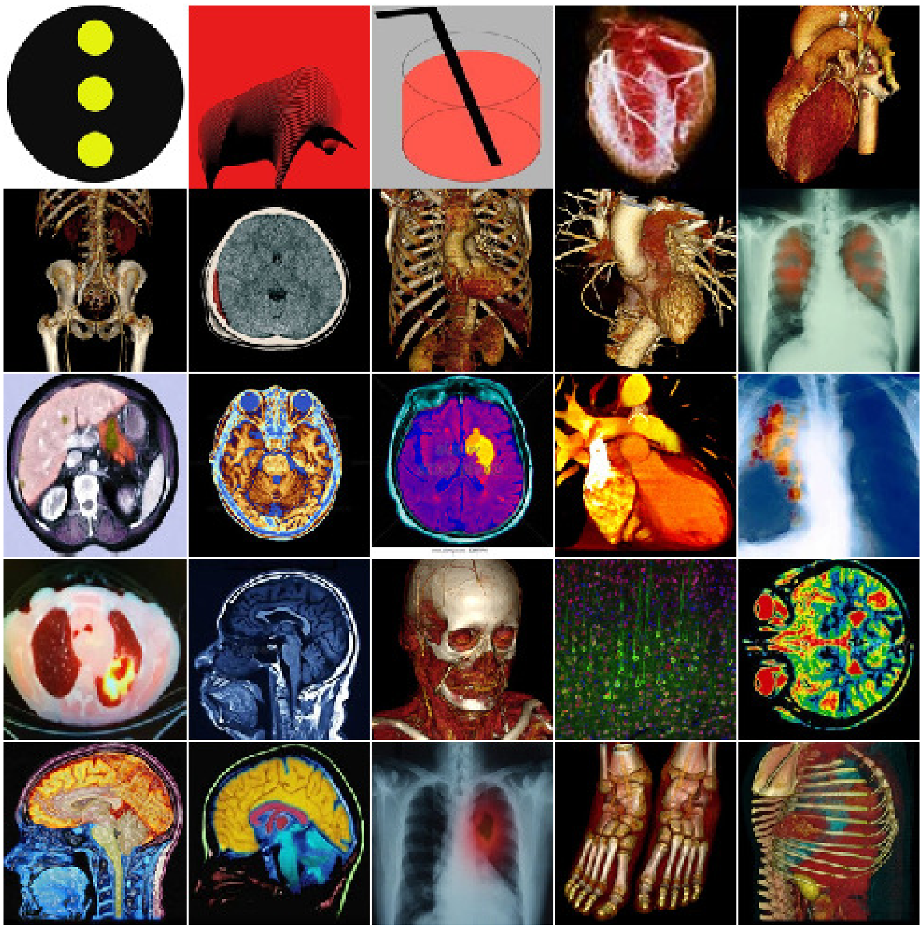


Figure 5.12: Decompressed image list.

$2.746 \times 10^{-6} O(P + |\beta| * log|\beta|)$, $1.171 \times 10^{-6} O(P)$ and $4.452 \times 10^{-5} O(|\beta| + P)$ for Run-length, Shannon–Fano, Huffman, LZW and Arithmetic coding, respectively, where P indicates the number of pixels and β represents the number of different pixels of an image.

5.3 Summary

This study presents a detailed analysis of some commonly used entropy coding techniques, such as run-length, Shannon–Fano, Huffman, LZW, and arithmetic coding. The relevance of these techniques comes from the fact that most of the other recently advanced lossless or lossy algorithms use one of them as a part of their compression procedure. All the mentioned algorithms have been discussed using a common numeric data set. Later, both computer-generated and actual medical images are used to assess the efficiency of such entropy coding methods. We also used standard metrics such as encoding time, decoding time, average code length, compression ratio, efficiency, and PSNR to measure the superiority of such techniques. Finally, we noticed that Huffman coding outperforms other entropy coding methods in the case of real-time lossless compression applications.

Chapter 6

PCBMS: A Model to Select an Optimal Lossless Image Compression Technique

This chapter presents a parameter combination-based method selection (PCBMS) approach to select an optimal lossless data compression technique and provides an analysis based on experimental results to show its effectiveness. There are different types of data such as image, audio, video, and text. These data are classified based on the number of bits. Many algorithms have been developed to compress data over the past few decades, but no developed algorithm works well on all types of data. Lossless data compression techniques are mainly evaluated based on the compression ratio, encoding, and decoding time. While a higher compression ratio is more important for some applications, others may require faster encoding or decoding, or both. Alternatively, each of the three parameters can be equally significant. Choosing an optimal algorithm from many algorithms based on an application's requirements is a significant challenge. By analyzing the data from each perspective, this model recommends an algorithm as the best for each type of data. Based on the proposed model, an analysis is provided. For some sets of data, it has been demonstrated that the proposed method gives a better prediction to select an algorithm according to the needs of an application.

6.1 Introduction

As technology advances, the demand for data compression is increasing daily, and new applications are being developed in which data compression is more significant. Some state-of-the-art data compression algorithms provide higher compression ratios, but require longer times for encoding, decoding, or both. Meanwhile, a fast algorithm cannot compress more [29]. The same situation occurs in the case of machine or deep learning-based lossless data compression models.

Schiopu et al. proposed a deep learning-based lossless image coding technique named CBPNN (an integration of the IResLNN predictor and the context-based bit-plane codec (CBP) [165] and compared it to lossless HEVCIntra [166], lossless JPEG (JPEG-LS) [87], context-based, adaptive, lossless image codec (CALIC) [167], free lossless image format (FLIF) [109], and MP-CNN [168]. This article shows that CBPNN outperforms 4.5% over REP-CNN [169] for the EPFL dataset [36] and 13.7%, 35.4%, 31.3%, and 10.6% over MP-CNN, JPEG-LS, CALIC, and FLIF, respectively, for the UVG-TUT dataset [37]. However, the High Efficiency Video Coding (HEVC) standard outperformed 20.12% for the video frames on the UVG-TUT dataset on average. Although speed is an essential factor of data compression, the article did not discuss the encoding or decoding speeds of the algorithms.

Rhee et al. [170] proposed a lossless image compression technique based on the learning of pixel values and contexts through multilayer perceptrons (MLPs) and compared various types of nonlearning based codecs (Better Portable Graphics (BPG) [171], Portable Network Graphics (PNG) [172], JPEG-LS [87], JPEG 2000 [121], LCIC [173], WebP [174], FLIF [109], JPEG-XL [175]) and learning-based (L3C [3], CBPNN [165], CWPLIC [176], PixelCNN [177], MS-PixelCNN [178], IDF [179]) codecs on different datasets. For the 4KUHD grayscale dataset, the proposed method provided better performance in terms of bits per pixel (bpp) than BPG, PNG, JPEG-LS, JPEG 2000, LCIC, WebP, FLIF, JPEG-XL and CWPLIC, except for CBPNN. However, CBPNN requires more time. Although CBPNN results in approximately 31.28% more compression, it requires 85% more time than the proposed method. For the ImageNet 64x64 dataset, only PixelCNN, MS-PixelCNN, and integer discrete flow (IDF) require 26.24%, 21.8%, and 15.56% less storage but are 99.92%, 91.7%, and 77.97% slower, respectively, than the proposed method. The method also provided better performance for the FLICKR2K and DIV2K datasets. The article also showed a comparison to the classic dataset. FLIF and the proposed method give the lowest 11.10 and 4.25 bpp for the color and grayscale images in the dataset, respectively. However, the proposed method was slower than the nonlearning-based techniques.

Ma et al. [180] proposed an optimized image compression scheme called iWave++, which supports lossy and lossless compression, where iWave++ is a trained wavelet-like transform technique. This article compared two deep learning-based methods (L3C [3] and Gated PixelCNN [177]) and four classical techniques (PNG, JPEG 2000, WebP, and FLIF) on two different datasets (ImageNet32 and ImageNet64) based on bpp, and the results showed that iWave++ outperformed all methods except Gated PixelCNN. However, Gated PixelCNN had more parameters than iWave++. In the article, the authors also showed another version of iWave++, called Universal iWave++, and gave a comparison based on the results of the grayscale Kodak dataset [108]. The outcomes showed that Universal iWave++ outperformed all methods (JPEG 2000, WebP, and FLIF) in terms of bpp.

Salimans et al. proposed PixelCNN++ [181], an updated version of PixelCNN, and showed that PixelCNN++ provided a better compression performance than Deep Diffusion [182], Nonlinear independent components estimation (NICE) [183], DRAW [184], Deep GMMs [185], Conv Deep Recurrent Attentive Writer (DRAW) [186], real-valued nonvolume preserving (Real NVP) [187], PixelCNN [188], VAE with inverse autoregressive flow (IAF) [189], Gated PixelCNN [177], and PixelRNN [190] on the CIFAR-10 dataset.

Ionut et al. proposed an improved CNN-based intraprediction method (AP-CNN) for lossless video coding applications [191]. The authors explained that CNN-based data compression techniques required a considerable amount of time for compression. They also showed that the proposed method provided 5% and 5.8% better reduction than Lossless HEVCIntra [166] on the HEVC-vTSEQ and TUT-vTSEQ datasets, respectively, and was approximately 97.71% faster than the CNN-based technique for the Basket-ballPass video sequence.

BPG is a lossy image compression technique developed based on the HEVC video coding standard [166]. In their article, Mentzer et al. [192] leveraged the BPG and a convolutional neural network-based probabilistic model to develop a lossless image compression technique and compared it with L3C, PNG, JPEG 2000, WebP, and FLIF based on four datasets: Open Images [193], CLIC.mobile, CLIC.pro [66], and DIV2K [194]. In terms of bits per subpixel (bpsp), the authors showed that their approach continuously outperformed L3C, PNG, WebP, and JPEG 2000 and provided a better outcome than FLIF for the Open Image dataset.

All published research works compare the state-of-the-art techniques based on the bpp, encoding time or decoding time [3,87,109,121,165–192,195–197]. In this type of comparison, we can easily determine which algorithm is better based on which parameter. For example, if we have two algorithms, A1 and A2, A1 may provide more compression than A2 but takes longer for encoding and decoding. In this case, we cannot easily state that A1 is better because the actual performance of a lossless data compression algorithm depends on all parameters (compression ratio, encoding time, and decoding time). For some applications, we must choose an algorithm that provides better performance based on a combination of any two or three evaluation parameters, which is a major challenge. In addition, based on the requirement of an application, finding an optimal lossless data compression algorithm for each type of data is a significant challenge. There are various types of data, such as eight-bit and 16-bit grayscale and RGB (red, green, blue) images, binary images, indexed images, audio, video, text, and different algorithms show different performances for different data. To solve these issues, a mathematical theory is proposed in this article. An analysis based on some experimental results is also provided.

The organization of this publication is as follows. In Section 6.2, we present the motivation for the research work and define the problems. Then, in Section 6.3, the proposed model is presented, which can assist in selecting a better lossless data compression algorithm based on the demand of a user. Finally, Section 6.4 provides an analysis based on experimental results, and Section 6.6 presents the study conclusions.

6.2 Motivation and Problem Formulation

Day-by-day development of a learning-based data compression technique is increasing. Additionally, nonlearning methods such as FLIF, Avif, and WebP are currently used in many applications. For example, if a dataset contains different types of images, various learning and nonlearning lossless image compression techniques can be applied. Among the state-of-the-art techniques, a better technique based on the compression ratio, encoding time, or decoding time can be easily found. With the advancement of technology, users' requirements are increasing. Various types of applications are being developed based on different conditions. Some applications may require a higher compression ratio and lower encoding time, lower encoding and decoding time, or higher compression ratio and lower decoding time. Alternatively, all three parameters are equally important. It has been observed that the method or learning-based model that provides a better compression ratio takes a longer time to encode or decode. Some techniques require more time for encoding than for decoding. Therefore, the most important questions are: 1. which technique is better when any two of the parameters (compression ratio, encoding, and decoding times) are equally significant for an application, or when all parameters are equally valuable?; 2. is there any such technique that works well for all types of data? Therefore, which technique is better for eight-bit RGB, eight-bit grayscale, 16-bit RGB, 16-bit grayscale or binary or indexed images or text, among others? For example, Table 4 from an article published in Computer Vision and Pattern Recognition (CVPR) 2019 [3] is depicted in Table 6.1. This table shows a comparison of the average encoding time, decoding time, and bpsp for 512 x 512 crops from the DIV2K dataset. Based on the results, the authors claim that their proposed method (L3C) outperforms WebP, JPEG 2000, and PNG on the DIV2K dataset. The table also shows that FLIF provides 2.887% more compression than L3C. We can agree with the authors that L3C outperforms PNG, JPEG 2000, and WebP in terms of bpsp; however, the encoding and decoding procedures of L3C are very slow in comparison to all other methods. Although FLIF outperformed all methods in terms of bpsp, it is approximately seven times slower in encoding and slightly faster in decoding than L3C. From Table 6.1, we can quickly determine that JPEG 2000 and PNG are better for encoding and decoding times, respectively. Based on any two or all parameters from Table 6.1, we cannot determine whether L3C is better because it requires more time to encode and decode. It continues to be difficult to state which method actually works well for images (a graphical representation of Table 6.1 is shown in Figure 6.1 for quick understanding). How do we find a good algorithm in these cases? This research article proposes a straightforward but innovative technique to choose a better lossless data compression algorithm based on all parameters or any combination of them.

Table 6.1: Comparison of the average bpsp, encoding and decoding times on the DIV2K dataset [3]

Codec	Encoding (s)	Decoding (s)	bpsp
L3C	0.242	0.374	3.386
PNG	0.213	6.09×10^{-5}	4.733
JPEG 2000	1.48×10^{-2}	2.26×10^{-4}	3.471
WebP	0.157	7.12×10^{-2}	3.447
FLIF	1.72	0.133	3.291

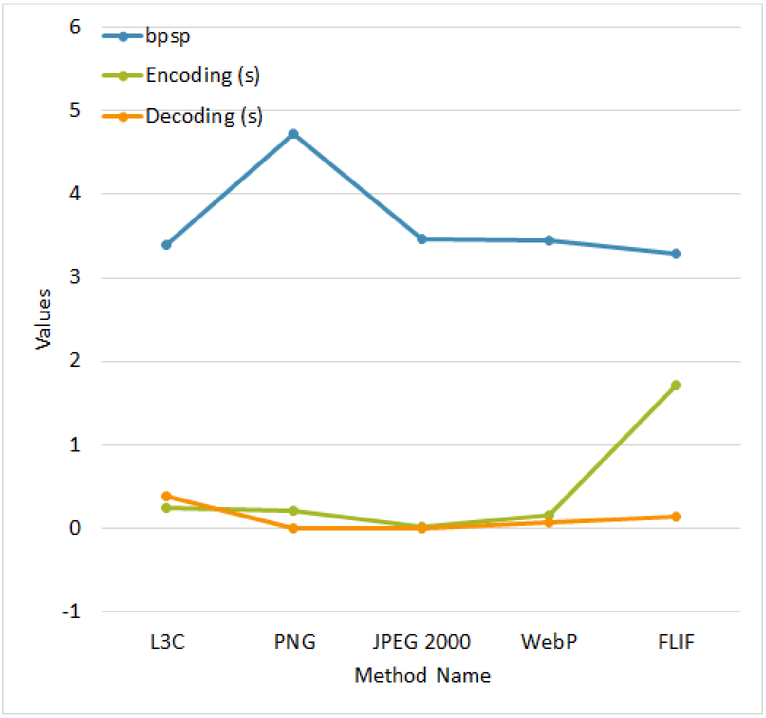


Figure 6.1: The graphical representation of Table 6.1

6.3 Proposed Model

Suppose that we have N images $(I_1, I_2, ..., I_N)$, apply P methods $(M_1, M_2, ..., M_P)$ on them, and obtain the result of $(E_{1,i}, E_{2,i}, ..., E_{N,i})$, $(D_{1,i}, D_{2,i}, ..., D_{N,i})$, and $(BPI_{1,i}, BPI_{2,i}, ..., BPI_{N,i})$ as the encoding time, decoding time, and bits per item for each image, respectively, for the i^{th} method. Similarly, we obtain the encoding time, decoding time, and bits per item for each method. Here, BPI is the number of bits per pixel (bpp), bits per subpixel (bpsp), or bits per symbol (bps). The average encoding time (AET), average decoding time (ADT), and average bits per item (ABPI) are calculated using Algorithm 3 for the i^{th} method:

Better methods can be found for each individual parameter using Algorithm 4. The best method in terms of encoding time (BM_{ET}), best method in terms of decoding time (BM_{DT}), and best method in terms of bits per item (BM_{BPI}) represent a better method in the case of encoding time, decoding time, and bits per item, respectively, for images $(I_1, I_2, I_3, ..., I_N)$.

To calculate the performance of the algorithms based on all parameters, all values must be converted to a space in which all values are measured based on a specific value (for example, 1). As a result, in the case of a particular parameter, it is easy to observe the effect of an algorithm on the results of all methods. Thus, we converted the values obtained from Algorithm 3 to another space using Algorithm 5, where BET_j, BDT_j, and $BBPI_j$ represent the balanced encoding

Algorithm 3: Average values

```
1  $i \leftarrow 1$;
2  while $i \leq P$ do
3      $AET_i \leftarrow \frac{1}{N}(E_{1,i} + E_{2,i} + E_{3,i} + ... + E_{N,i})$;
4      $ADT_i \leftarrow \frac{1}{N}(D_{1,i} + D_{2,i} + D_{3,i} + ... + D_{N,i})$;
5      $ABPI_i \leftarrow \frac{1}{N}(BPI_{1,i} + BPI_{2,i} + ... + BPI_{N,i})$;
6      $i \leftarrow i + 1$;
7  end
```

Algorithm 4: The best method selection approach

```
1  Individual_Method Selection(AET, ADT, ABPI, M);
2  $Min_{AET}$, $Min_{ADT}$, and $Min_{ABPI} \leftarrow$ a big value;
3  i ← 1;
4  while $i \leq P$ do
5      if $AET[i] < Min_{AET}$ then
6          $Min_{AET} \leftarrow AET[i]$;
7          $BM_{ET} \leftarrow M_i$;
8      end
9      if $ADT[i] < Min_{ADT}$ then
10         $Min_{ADT} \leftarrow ADT[i]$;
11         $BM_{DT} \leftarrow M_i$;
12     end
13     if $ABPI[i] < Min_{ABPI}$ then
14         $Min_{ABPI} \leftarrow ABPI[i]$;
15         $BM_{BPI} \leftarrow M_i$;
16     end
17     i ← i+1;
18 end
19 return $(BM_{ET}, BM_{DT}, BM_{BPI})$ ;
```

time, balanced decoding time and balanced bits per item, respectively, for the j^{th} method. The goodness of any lossless data compression algorithm depends on lower encoding time, lower decoding time, and lower BPI. In Algorithm 5, a lower value for each parameter indicates the higher performance of an algorithm:

Algorithm 5: Balanced values

```
1 $j = 1$;
2 while $j \leq P$ do
3     $BET_j = \frac{AET_j}{\sum_{i=1}^{P} AET_i}$;
4     $BDT_j = \frac{ADT_j}{\sum_{i=1}^{P} ADT_i}$;
5     $BBPI_j = \frac{ABPI_j}{\sum_{i=1}^{P} ABPI_i}$;
6     $j = j + 1$
7 end
```

Then, we calculated the overall impact of each algorithm for each combination of parameters and selected an algorithm as the best for each category using Algorithm 6. In the algorithm, IED_K, $IEBPI_K$, $IDBPI_K$, and $IEDBPI_K$ indicate the overall impact of the K^{th} algorithm in terms of a combination of encoding and decoding times, encoding time and bits per item, decoding time and bits per item, and all parameters, respectively.

6.4 Experimental Results and Analysis

This section analyzes recently published results in well-known journals and conferences. In all figures in this article, ET and DT represent the encoding time and decoding time, respectively. The outcomes of the proposed method in Table 6.1 are shown in Figure 6.2. Mentzer et al. [3] published Table 6.1 in the CVPR 2019. Figure 6.2 shows that JPEG 2000 provides the highest (40.6%, 31.8%, 32.9%, and 88.9%) performances for "bpsp+ET+DT", "bpsp+ET", "bpsp+DT", and "ET+DT", respectively. According to Table 6.1, FLIF performed well in terms of bpsp; however, the method provided the lowest performances (7.0%, 6.7%, and 0.6%) for all cases except "bpsp+DT", as shown in Figure 6.2. For "bpsp+DT", L3C provided the lowest (7.5%) performance. Therefore, JPEG 2000 is better for all of these cases.

Cao et al. [4] compared some classical and learning-based methods on two datasets (ImageNet64 [198] & Open Images [193]) as demonstrated in Table 6.2. This table shows that PNG outperforms both datasets in terms of encoding and decoding speed. However, in terms of bpsp, IDF and SReC provide better results for ImageNet64 and Open Images, respectively. The results of our proposed method are demonstrated in Figure 6.3 and Figure 6.4 for ImageNet64 and Open Images of Table 6.2, respectively. Figure 6.3 shows that WebP provided the highest (21.9% and 21.5%) performances for the combinations "bpsp+ET+DT" and "bpsp+DT", respectively, for the ImageNet64 dataset. However, for the combinations "bpsp+ET" and "ET+DT", FLIF and PNG gave the highest (20.1% and 87.7%) performances. Meanwhile, for the Open Images dataset, Figure 6.4 shows that WebP, SReC, FLIF, and PNG provided the highest (21.7%, 21.5%, 21.2%, and 59.1%) performances, for the combinations "bpsp+ET+DT", "bpsp+ET", "bpsp+DT", and "ET+DT", respectively.

6.5 Tests to confirm the authenticity of the proposed method

We have provided evidence to confirm the veracity of the proposed method. Tables 6.3-6.6 present some random values for the methods (M1, M2, M3, ..., M15). We intentionally set low

Algorithm 6: The best method selection approach based on any combination

```
1  SelectedMethod Selection(BET, BDT, BBPI, M);
2  $C = 1$;
3  $K = 1$;
4  while $C \leq P$ do
5      $A1[C] = BET[C] + BDT[C]$;
6      $A2[C] = BET[C] + BBPI[C]$;
7      $A3[C] = BDT[C] + BBPI[C]$;
8      $A4[C] = BET[C] + BDT[C] + BBPI[C]$;
9      $C = C + 1$;
10 end
11 while $K \leq P$ do
12     $IED_K = \frac{1}{A1[K] \times \sum_{C=1}^{P} \frac{1}{A1[C]}} \times 100$;
13     $IEBPI_K = \frac{1}{A2[K] \times \sum_{C=1}^{P} \frac{1}{A2[C]}} \times 100$;
14     $IDBPI_K = \frac{1}{A3[K] \times \sum_{C=1}^{P} \frac{1}{A3[C]}} \times 100$;
15     $IEDBPI_K = \frac{1}{A4[K] \times \sum_{C=1}^{P} \frac{1}{A4[C]}} \times 100$;
16     $K = K + 1$;
17 end
18 Set the index of the highest value of $IED_K$, $IEBPI_K$, $IDBPI_K$, and $IEDBPI_K$ to
     $ILV$;
19 Return $M[ILV[1]]$, $M[ILV[2]]$, $M[ILV[3]]$, and $M[ILV[4]]$ that represent a method
     as the best in terms of a combination of encoding and decoding times, encoding time
     and bits per item, decoding time and bits per item, and all parameters, respectively.
```

Table 6.2: Comparison of the average bpsp, encoding and decoding times on ImageNet64 and Open Images datasets [4]

	ImageNet64			Open Images		
Method	Encoding (s)	Decoding (s)	bpsp	Encoding (s)	Decoding (s)	bpsp
PNG [99]	1.3×10^{-3}	8.0×10^{-5}	5.74	0.17	9.8×10^{-5}	4.03
WebP [103]	0.021	2.1×10^{-4}	4.64	0.4	7.0×10^{-4}	3.03
FLIF [109]	0.022	0.01	4.54	1.23	0.3	2.87
L3C [3]	0.031	0.023	4.42	1.33	1.13	2.99
IDF [179]	1.33	1.02	3.9	57.31	62.33	2.76
SReC	0.044	0.071	4.29	0.99	1.15	2.7
MIN	1.3×10^{-3}	8.0×10^{-5}	3.9	0.17	9.8×10^{-5}	2.7

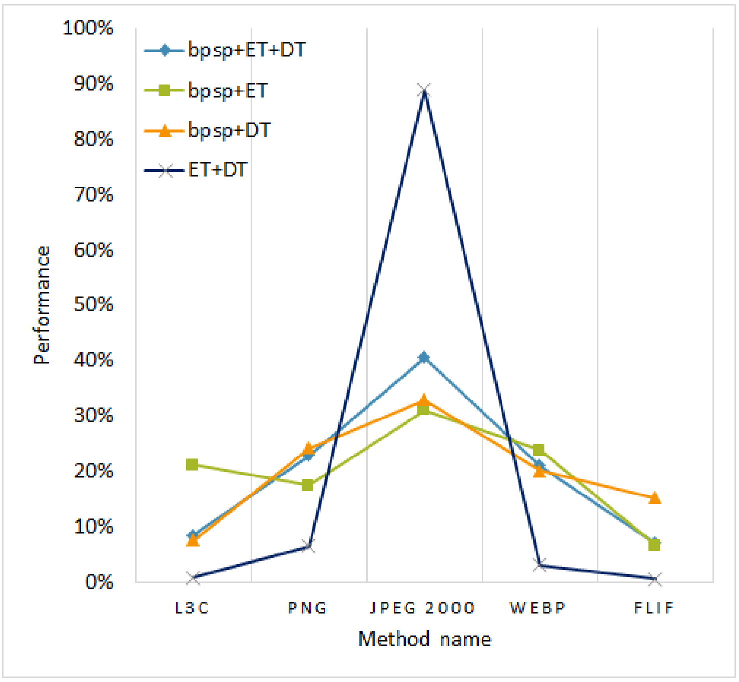


Figure 6.2: The outcomes of the proposed method for the Table 6.1

values for methods M10, M2, M7, and M5 in Tables 6.3, 6.4, 6.5, and 6.6 for the encoding and decoding time, encoding time and bps, decoding time and bps, and all parameters, respectively. Based on the proposed method, the outcomes are demonstrated in Figures 6.5-6.8 for Tables 6.3-6.6, respectively. Figures 6.5-6.8 show the highest performances (16.2%, 13.8%, 11.1%, and 15.5%) for methods M10, M2, M7, and M5, respectively. Therefore, the results show that the proposed method perfectly predicts the best technique of many methods for any combination of parameters.

6.6 Conclusions

A mathematical model to select an optimal lossless image compression technique is proposed in this article. This study shows that each algorithm was evaluated based on a specific parameter in each research work. However, the performance of a lossless image compression algorithm depends on all parameters (bpp, encoding and decoding time) and does not singly depend on any of them. Therefore, the proposed method predicts a better lossless image compression algorithm for any combination of parameters and provides the actual impact of each compared algorithm as a percentage.

Experimental results show that the proposed method perfectly predicts a better lossless image compression method for any combination of parameters. The results in this article show that although learning-based methods provide a good bpsp, the classical methods are better for lossless image compression overall.

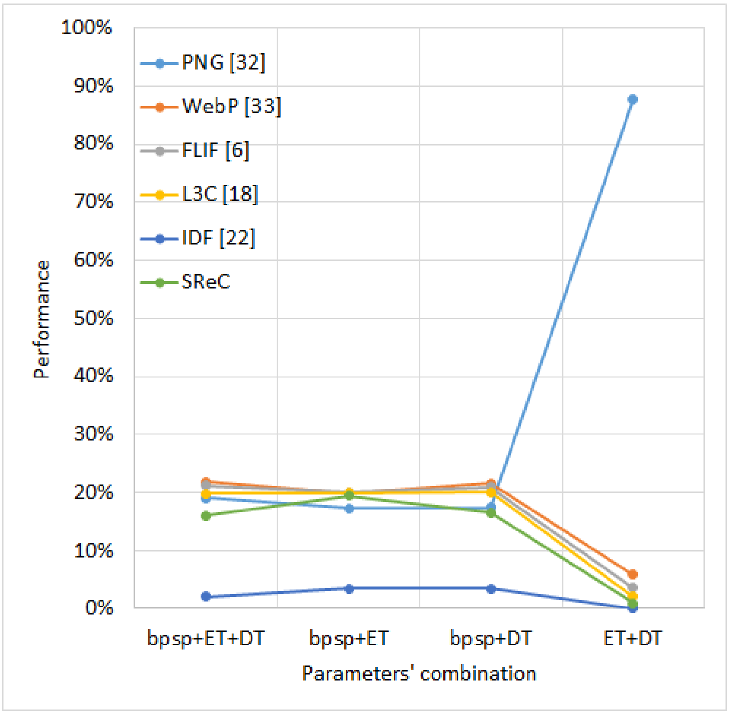


Figure 6.3: The outcomes of the proposed method for the ImageNet64 dataset of the Table 6.2

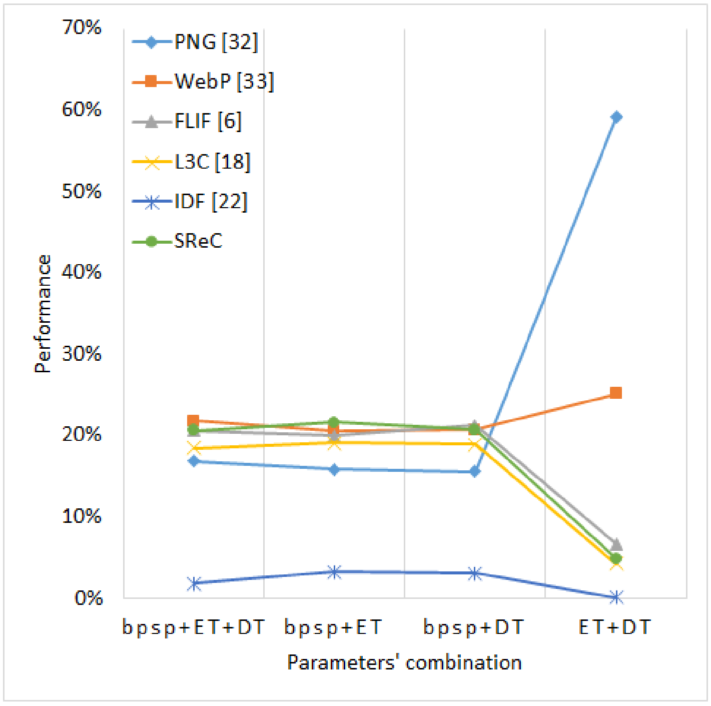


Figure 6.4: The outcomes of the proposed method for the Open Images dataset of the Table 6.2

Table 6.3: Random dataset 1

Method	bpsp	Encoding (s)	Decoding (s)
M1	4.202	0.943	0.297
M2	4.871	1.765	0.776
M3	2.087	1.357	0.579
M4	2.577	1.008	1.101
M5	4.111	0.918	1.182
M6	4.721	1.496	0.609
M7	2.568	0.893	0.998
M8	2.084	1.467	0.769
M9	2.391	0.694	0.488
M10	4.331	**0.626**	**0.125**
M11	5.959	0.882	1.347
M12	5.501	0.938	0.217
M13	2.519	1.809	0.979
M14	4.936	1.535	1.439
M15	5.158	1.161	0.643
MIN	**2.084**	**0.626**	**0.125**

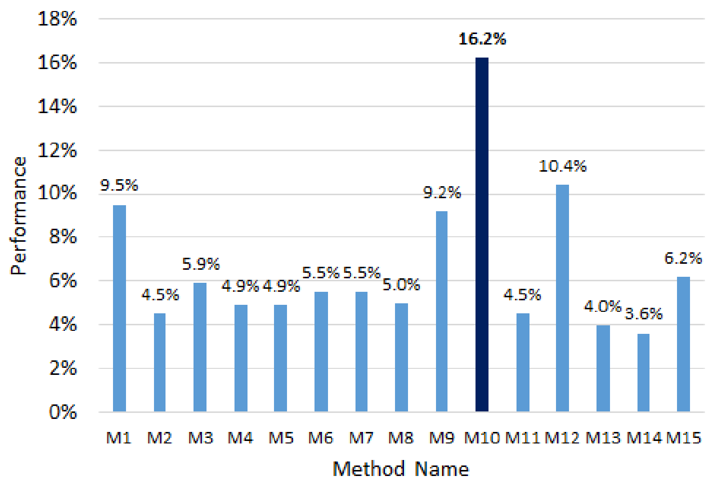


Figure 6.5: The outcomes of the proposed method for the Table 6.3

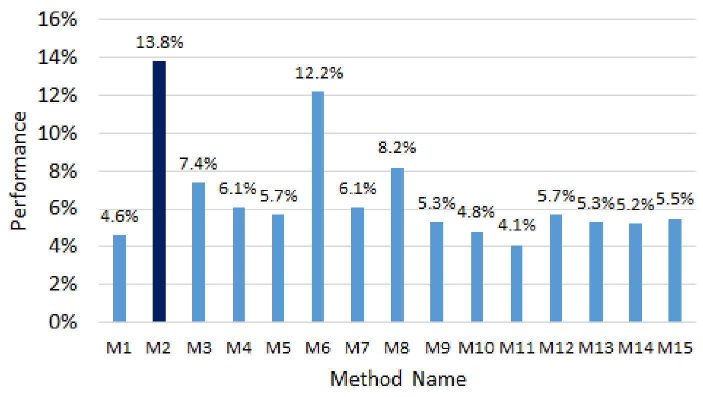


Figure 6.6: The outcomes of the proposed method for the Table 6.4

Table 6.4: Random dataset 2

Method	**bpsp**	**Encoding (s)**	**Decoding (s)**
M1	5.243	1.086	1.184
M2	**2.231**	**0.244**	1.236
M3	3.134	0.724	0.706
M4	5.336	0.484	0.951
M5	2.39	1.37	1.355
M6	2.25	0.345	0.835
M7	4.933	0.601	1.42
M8	3.115	0.577	1.006
M9	3.495	1.228	0.499
M10	4.797	1.109	0.685
M11	4.796	1.545	0.957
M12	3.904	0.971	1.121
M13	4.693	0.915	0.403
M14	2.36	1.554	0.099
M15	2.979	1.279	1.124
MIN	**2.231**	**0.244**	**0.099**

Table 6.5: Random dataset 3

Method	**bpsp**	**Encoding (s)**	**Decoding (s)**
M1	2.996	0.698	1.461
M2	2.55	0.825	1.875
M3	2.702	0.878	1.189
M4	3.907	1.578	1.123
M5	5.685	1.015	1.664
M6	4.569	0.938	1.557
M7	**2.543**	0.727	**0.666**
M8	3.278	1.118	1.031
M9	5.037	1.088	1.658
M10	5.699	1.381	1.206
M11	4.525	1.263	1.313
M12	4.634	1.897	1.758
M13	5.507	1.179	1.125
M14	5.002	0.725	0.801
M15	2.971	0.801	0.896
MIN	**2.543**	**0.698**	**0.666**

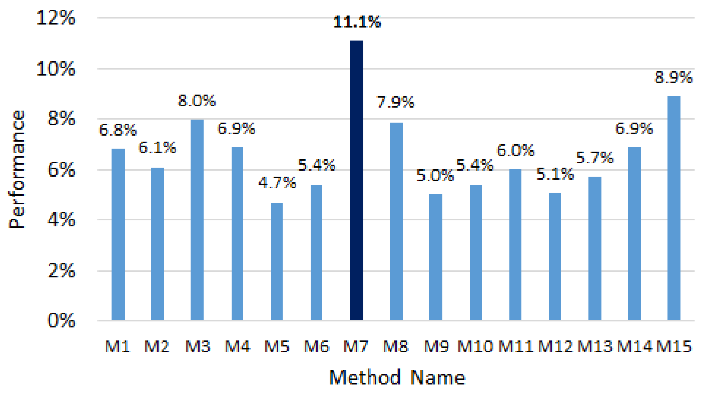


Figure 6.7: The outcomes of the proposed method for the Table 6.5

Table 6.6: Random dataset 4

Method	bpsp	Encoding (s)	Decoding (s)
M1	3.086	1.341	1.009
M2	3.909	0.811	1.125
M3	5.193	0.838	1.414
M4	4.972	1.493	0.37
M5	**2.498**	**0.288**	**0.189**
M6	2.77	1.488	0.318
M7	4.047	0.778	1.402
M8	5.726	1.049	0.711
M9	2.776	0.868	0.818
M10	2.683	1.234	1.178
M11	3.865	0.296	0.615
M12	2.941	1.509	1.314
M13	2.988	1.176	1.363
M14	3.963	1.298	1.797
M15	2.526	1.099	0.464
MIN	**2.498**	**0.288**	**0.189**

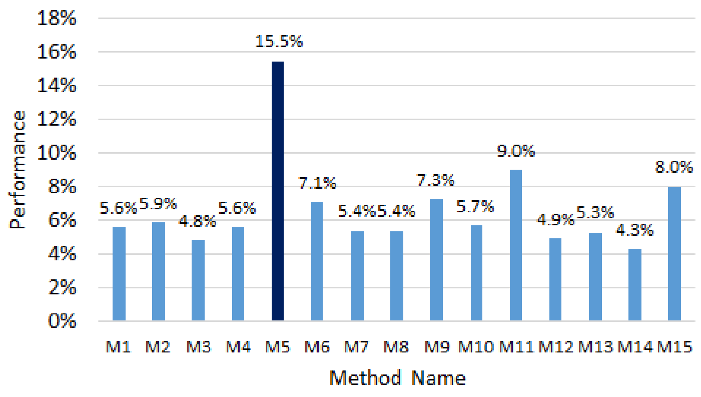


Figure 6.8: The outcomes of the proposed method for the Table 6.6

Chapter 7

The Impact of State-of-the-Art Techniques for Lossless Still Image Compression

A great deal of information is produced daily, due to advances in telecommunication, and the issue of storing it on digital devices or transmitting it over the Internet is challenging. Data compression is essential in managing this information well. Therefore, research on data compression has become a topic of great interest to researchers, and the number of applications in this area is increasing. Over the last few decades, international organizations have developed many strategies for data compression, and there is no specific algorithm that works well on all types of data. The compression ratio, as well as encoding and decoding times, are mainly used to evaluate an algorithm for lossless image compression. However, although the compression ratio is more significant for some applications, others may require higher encoding or decoding speeds or both; alternatively, all three parameters may be equally important. The main aim of this research is to analyze the most advanced lossless image compression algorithms from each point of view and evaluate the strength of each algorithm for each kind of image. We develop a technique regarding how to evaluate an image compression algorithm that is based on more than one parameter. The findings that are presented in this research may be helpful to new researchers and to users in this area.

7.1 Introduction

A huge amount of data is now produced daily, especially in medical centres and on social media. It is not easy to manage these increasing quantities of information, which are impractical to store and take a huge amount of time to transmit over the Internet. According to the sixth edition of a report by DOMO [199], more than 2.5 quintillion bytes of data are produced daily, and this figure is growing. The report further estimates that approximately 90% of the world's data were produced between 2018 and 2019, and that each person on earth will create 1.7 MB of data per second by 2020. As a result, storing large amounts of data on digital devices and quickly transferring them across networks is a significant challenge. There are three possible solutions to this problem: better hardware, better software, or a combination of both. However, so much information is being created that it is almost impossible to design new hardware that is sufficiently competitive, due to the many limitations on the construction of hardware, as reported in [200]. The development of better software is therefore the only solution to the problem.

One solution from the software perspective is compression. Data compression is a way of representing data using fewer bits than the original, to reduce the consumption of storage and bandwidth and increase transmission speed over networks [27]. Data compression can be applied in many areas, such as audio, video, text and images, and can be classified into two categories: lossless and lossy [201]. In lossy compression, irrelevant and less significant data are removed permanently, whereas in lossless compression, every detail is preserved and only statistical redundancy is eliminated. In short, lossy compression allows for slight degradation in the data, while lossless methods perfectly reconstruct the data from its compressed form [69, 152, 156, 202]. There are many applications for lossless data compression techniques, such as medical imagery, digital radiography, scientific imaging, zip file compression, museums/art galleries, facsimile transmissions of bitonal images, business documents, machine vision, the storage and sending of thermal images taken by nano-satellites, observation of forest fires etc. [13, 32, 125, 203, 204]. In this article, we study the compression standards used for lossless image compression. There are many such methods, including run-length coding, Shannon–Fano coding, Lempel–Ziv–Welch (LZW) coding, Huffman coding, arithmetic coding, lossless JPEG, PNG, JPEG 2000, JPEG-LS, JPGE XR, CALIC , AVIF, WebP, FLIF, etc. However, we limit our analysis to lossless JPEG, PNG, JPEG 2000, JPEG-LS, JPGE XR, CALIC, AVIF, WebP and FLIF since these are the latest and most fully developed methods in this area. Although there are also many types of image, such as binary images, 8-bit and 16-bit grayscale images, 8-bit indexed images, and 8-bit and 16-bit RGB images, we cover only 8-bit and 16-bit grayscale and RGB images in this article. A detailed review of run-length, Shannon–Fano, LZW, Huffman and arithmetic coding was carried out in [27]. Four parameters are used to evaluate a lossless image compression algorithm: the compression ratio (CR), bits per pixel (bpp), encoding time (ET) and decoding time (DT), and bpp is the simply the inverse of the CR; we therefore consider only the CR, ET and DT when evaluating these methods. Most studies in this research area use only the CR to evaluate the effectiveness of an algorithm [205–208]. Although there are many applications for which higher CRs are important, others may require higher encoding or decoding speeds. Other applications may require high CR and low ET, low ET and DT, or high CR and decoding speed. In addition, there are also many applications where all three of these parameters are equally important. In view of this, we present an extensive analysis from each perspective, and examine the strength of each algorithm for each kind of image. We compare the performance of these methods based on public open datasets. More specifically, the main aim of this research is to address the following research issues:

RI1: How good is each algorithm in terms of the CR for 8-bit and 16-bit greyscale and RGB images?

RI2: How good is each algorithm in terms of the ET for 8-bit and 16-bit greyscale and RGB images?

RI3: How good is each algorithm in terms of the DT for 8-bit and 16-bit greyscale and RGB images?

RI4: How good is each algorithm in terms of the CR and ET for 8-bit and 16-bit greyscale and RGB images?

RI5: How good is each algorithm in terms of the CR and DT for 8-bit and 16-bit greyscale and RGB images?

RI6: How good is each algorithm in terms of the ET and DT for 8-bit and 16-bit greyscale and RGB images?

RI7: How good is each algorithm when all parameters are equally important for 8-bit and 16-bit greyscale and RGB images?

RI8: Which algorithms should be used for each kind of image?

The remainder of this research is structured as follows. Based on the usual parameters used to evaluate a lossless image compression technique, a detailed analysis of the experimental outcomes obtained using these algorithms is provided in section 7.2. In section 7.2, we define our proposed evaluation technique and give an extensive investigation based on the method. Finally, we conclude the paper in section 7.3.

7.2 Experimental Results and Analysis

The dependence of many applications on multimedia computing is growing rapidly, due to an increase in the use of digital imagery. As a result, the transmission, storage and effective use of images are becoming important issues. Raw image transmission is very slow, and gives rise to huge storage costs. Digital image compression is a way of converting an image into a format that can be transferred quickly and stored in a comparatively small space. In this paper, we provide a detailed analysis of the state-of-the-art lossless still image compression techniques. As we have seen, most previous survey papers on lossless image compression focus on the CR [19, 131, 156, 201, 202, 209–211]. Although this is an important measurement criterion, the ET and DT are two further important factors for lossless image compression. The key feature of this paper is that we not only carry out a comparison based on CR but also explore the effectiveness of each algorithm in terms of compressing an image based on several metrics. We use four types of images: 8-bit and 16-bit greyscale and RGB images. We applied state-of-the-art techniques to a total of nine uncompressed images of each type, taken from a database entitled "New Test Images - Image Compression Benchmark" [30].

Analysis based on usual parameters

In this section, we have analyzed based on every single parameter. Figures 7.1-7.3 show the 8-bit greyscale images and their respective ETs and DTs, and Table 7.1 shows the CRs of the images. Table 7.1 shows that FLIF gives the highest CR for the artificial.pgm, big_tree.pgm, bridge.pgm, cathedral.pgm, fireworks.pgm, and spider_web.pgm 8-bit greyscale images. For the rest of the images, AVIF provides the highest CR. JPEG XR gives the lowest CRs (3.196, 3.054 and 2.932) for three images (artificial, fireworks and flower_foveon, respectively), while for the rest of the images, lossless JPEG gives the lowest CR. In terms of the ET, Figure 7.4 shows that JPEG-LS gives the shortest times (0.046 and 0.115 s) for two images (artificial and fireworks) and JPEG XR gives the shortest times for the remainder of the images. FLIF takes the longest times (3.28, 15.52, 6.18, 2.91, 6.19, 2.73, 3.27 and 4.32 s, respectively) for all the images except flower_foveon.pgm. For this image, AVIF takes the longest time (1.54 s).

Figure 7.5 shows that at the decoding stage, CALIC takes the longest times (0.542, 0.89 and 0.898 s) for three images (fireworks.pgm, hdr.pgm, and spider_web.pgm, respectively), and WebP takes the longest times (3.26 and 2.33 s) for two images (big_tree.pgm and bridge.pgm, respectively), while FLIF takes the longest times for the rest of the images. On the other hand, PNG takes the shortest times, respectively, for all the images.

Examples of 8-bit RGB images are shown in Figure 7.4. Table 7.2 and Figures 7.5 and 7.6 show the values of CR, ET and DT for each of the images. From Table 6, it can be seen that FLIF give the highest CRs (18.446 and 5.742) for two images (artificial.PPM and fireworks.PPM, respectively), and AVIF gives the highest CRs for the rest of the images. JPEG XR gives the lowest CRs (4.316 and 3.121) for two images (artificial and fireworks, respectively), and for the remaining seven images, lossless JPEG gives the lowest CRs. Figure 7.5 shows that FLIF requires the longest ETs for all the images. JPEG XR gives the shortest ET for all images except artificial.pgm, for which JPEG-LS gives the shortest ET (0.16 s). For decoding, CALIC takes the longest time for all images except for bridge, cathedral, and deer, as shown in Figure 7.6, and FLIF takes the longest times (4.75, 2.379 and 5.75 s, respectively) for these images. Figure 7.6 also shows that JPEG XR takes the shortest time (0.157 s) to decode artificial.ppm, and PNG takes the shortest for the rest of the images.

Figures 7.7-7.9 and Table 7.3 show the 16-bit greyscale images and their respective ETs,

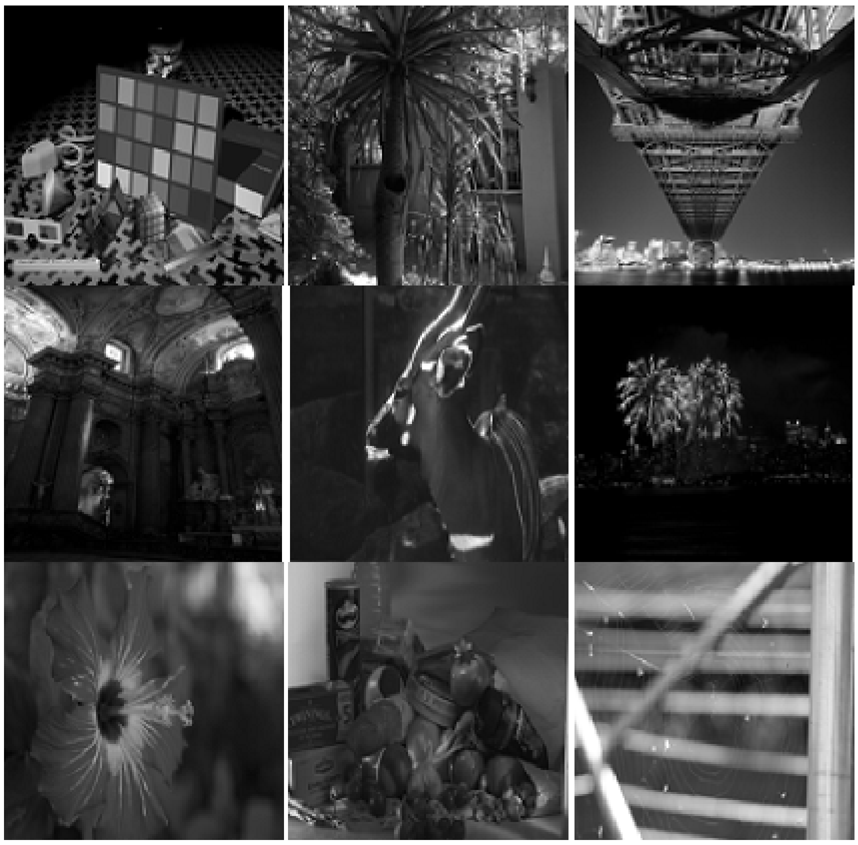
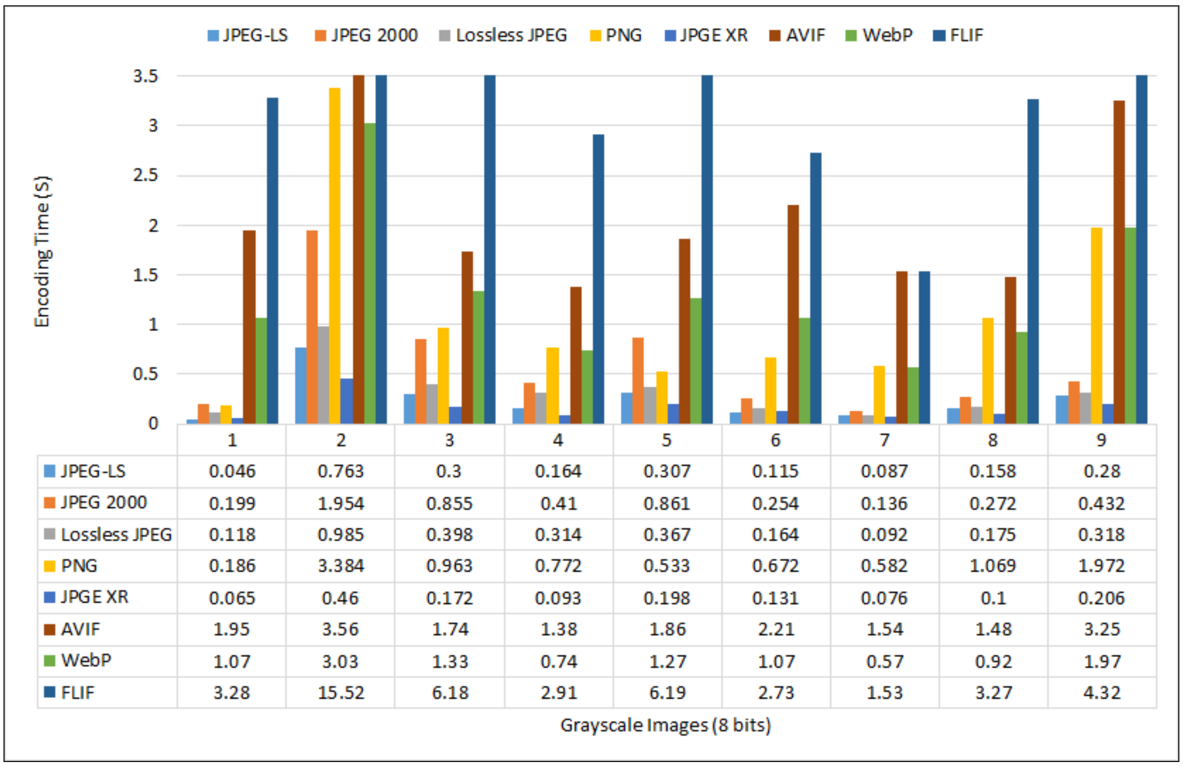


Figure 7.1: Examples of 8-bit greyscale images

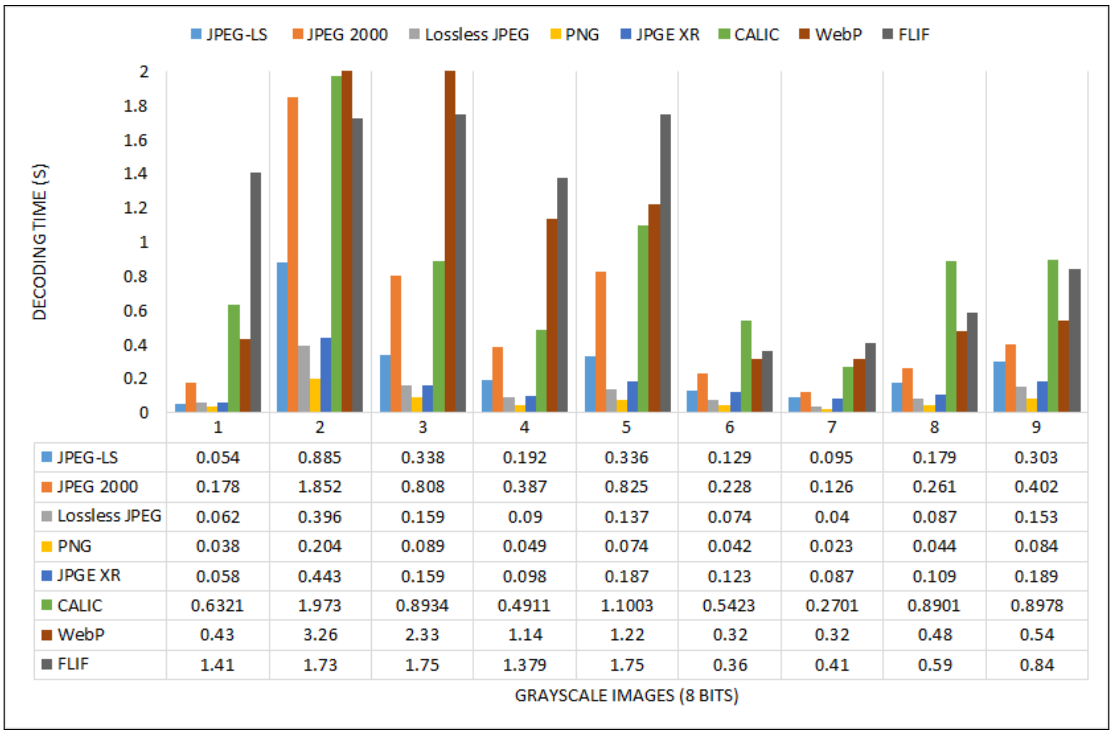
Table 7.1: Comparison of compression ratios for 8-bit greyscale images

Image Name (Dimensions)	**JPGE-LS**	**JPEG 2000**	**Lossless JPEG**	**PNG**	**JPEG XR**	**CALIC**	**AVIF**	**WebP**	**FLIF**
artificial.pgm (2048 × 3072)	10.030	6.720	4.884	8.678	3.196	10.898	4.145	10.370	13.260
big_tree.pgm (4550 × 6088)	2.144	2.106	1.806	1.973	2.011	2.159	2.175	2.145	2.263
bridge.pgm (4049 × 2749)	1.929	1.910	1.644	1.811	1.846	1.929	1.973	1.943	1.979
cathedral.pgm (3008 × 2000)	2.241	2.160	1.813	2.015	2.038	2.254	2.222	2.217	2.365
deer.pgm (2641 × 4043)	1.717	1.748	1.583	1.713	1.685	1.727	1.872	1.794	1.775
fireworks.pgm (2352 × 3136)	5.460	4.853	3.355	4.095	3.054	5.516	4.486	4.858	5.794
flower_foveon.pgm (1512 × 2268)	3.925	3.650	2.970	3.054	2.932	3.935	4.115	3.749	3.975
hdr.pgm (2048 × 3072)	3.678	3.421	2.795	2.857	2.847	3.718	3.825	3.456	3.77
spider_web.pgm (2848 × 4256)	4.531	4.202	3.145	3.366	3.167	4.824	4.682	4.133	4.876



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.046	0.763	0.3	0.164	0.307	0.115	0.087	0.158	0.28
JPEG 2000	0.199	1.954	0.855	0.41	0.861	0.254	0.136	0.272	0.432
Lossless JPEG	0.118	0.985	0.398	0.314	0.367	0.164	0.092	0.175	0.318
PNG	0.186	3.384	0.963	0.772	0.533	0.672	0.582	1.069	1.972
JPGE XR	0.065	0.46	0.172	0.093	0.198	0.131	0.076	0.1	0.206
AVIF	1.95	3.56	1.74	1.38	1.86	2.21	1.54	1.48	3.25
WebP	1.07	3.03	1.33	0.74	1.27	1.07	0.57	0.92	1.97
FLIF	3.28	15.52	6.18	2.91	6.19	2.73	1.53	3.27	4.32

Figure 7.2: Encoding times for 8-bit greyscale images



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.054	0.885	0.338	0.192	0.336	0.129	0.095	0.179	0.303
JPEG 2000	0.178	1.852	0.808	0.387	0.825	0.228	0.126	0.261	0.402
Lossless JPEG	0.062	0.396	0.159	0.09	0.137	0.074	0.04	0.087	0.153
PNG	0.038	0.204	0.089	0.049	0.074	0.042	0.023	0.044	0.084
JPGE XR	0.058	0.443	0.159	0.098	0.187	0.123	0.087	0.109	0.189
CALIC	0.6321	1.973	0.8934	0.4911	1.1003	0.5423	0.2701	0.8901	0.8978
WebP	0.43	3.26	2.33	1.14	1.22	0.32	0.32	0.48	0.54
FLIF	1.41	1.73	1.75	1.379	1.75	0.36	0.41	0.59	0.84

Figure 7.3: Decoding times for 8-bit greyscale images

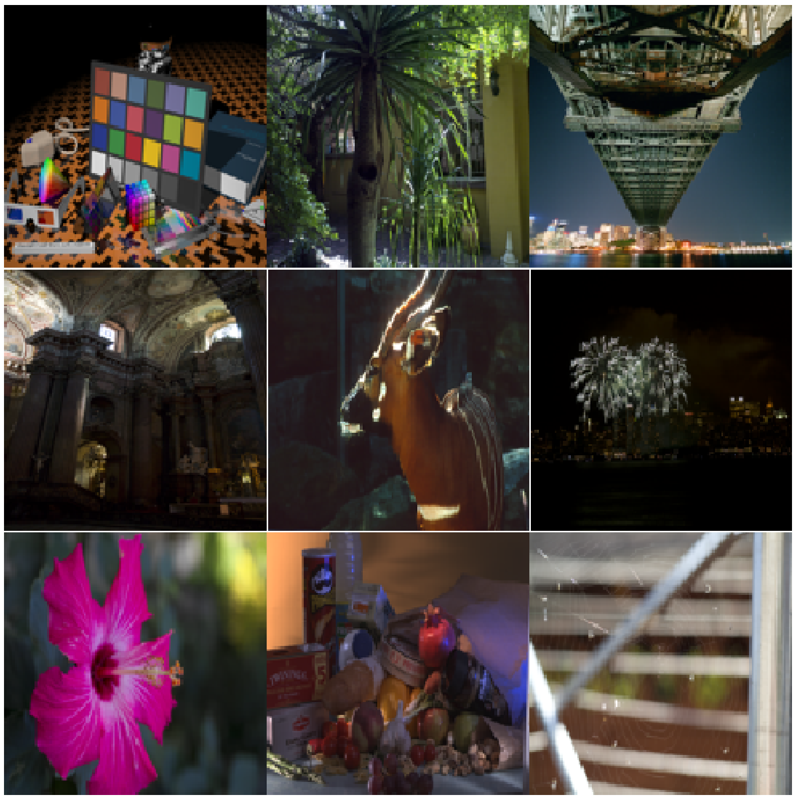
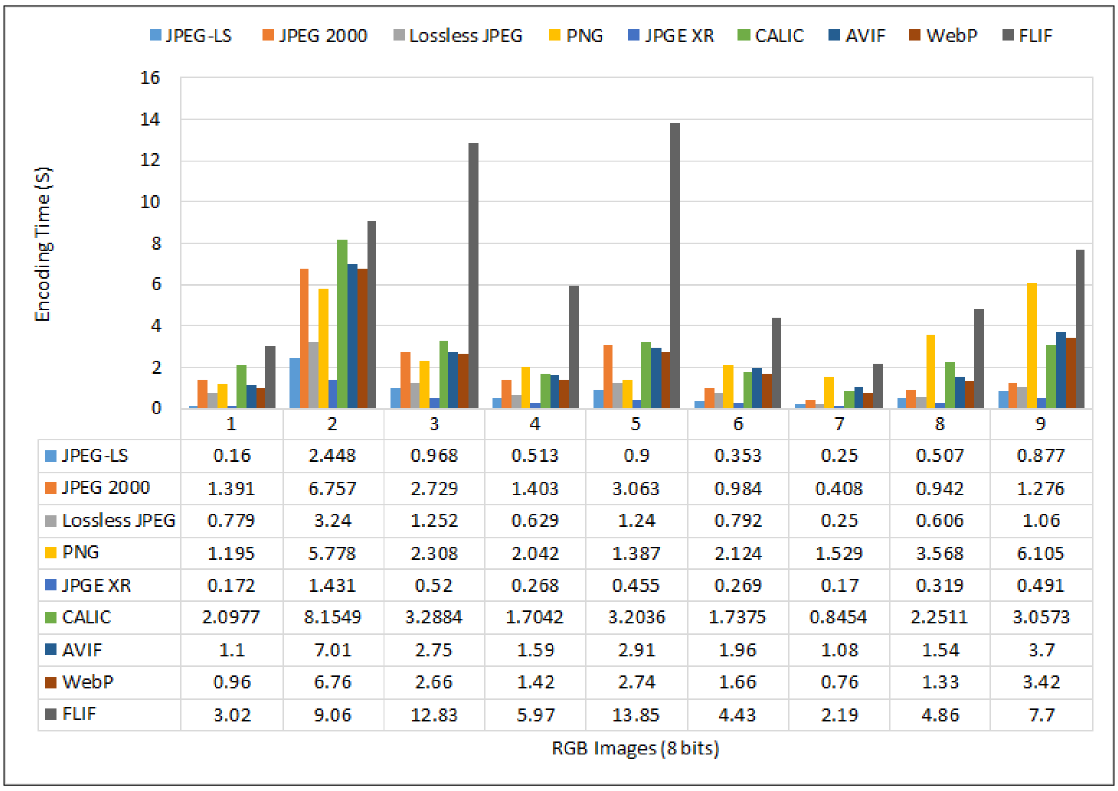


Figure 7.4: Examples of 8-bit RGB images

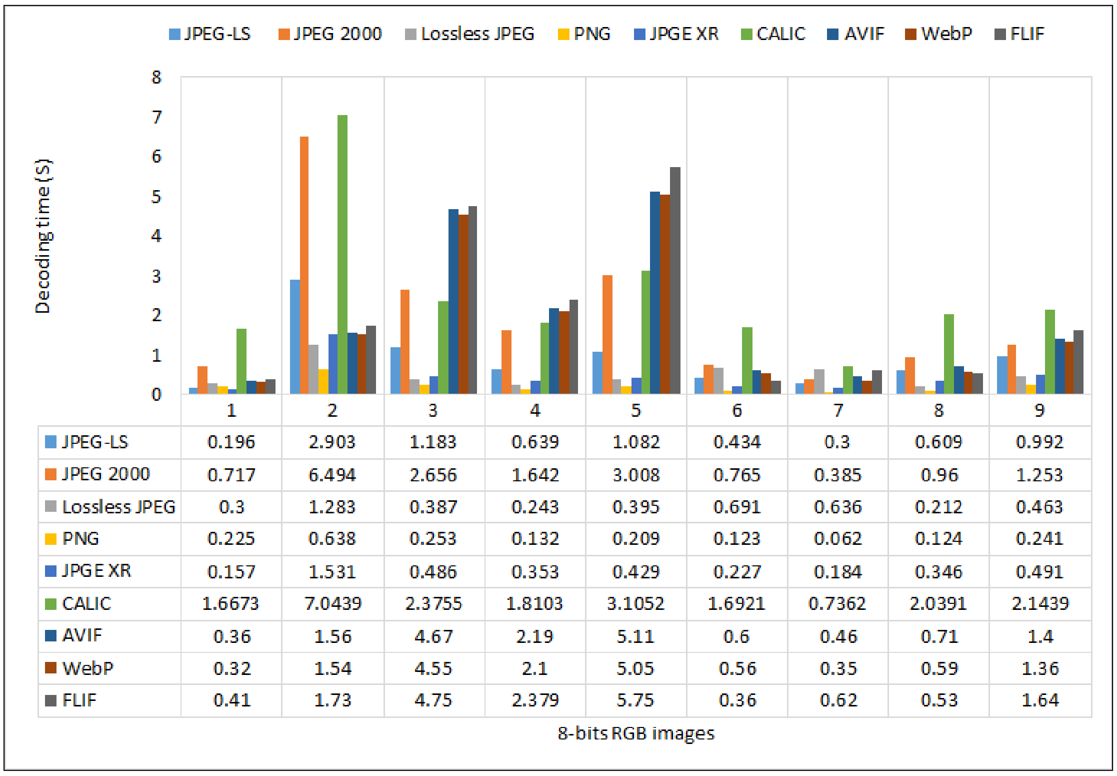
Table 7.2: Comparison of compression ratios for 8-bit RGB images

Image Name (Dimensions)	**JPGE-LS**	**JPEG 2000**	**Lossless JPEG**	**PNG**	**JPEG XR**	**CALIC**	**AVIF**	**WebP**	**FLIF**
artificial.pgm (2048 × 3072)	10.333	8.183	4.924	10.866	4.316	10.41	6.881	17.383	18.446
big_tree.pgm (4550 × 6088)	1.856	1.823	1.585	1.721	1.719	1.886	2.254	1.835	1.922
bridge.pgm (4049 × 2749)	1.767	1.765	1.553	1.686	1.735	1.784	2.21	1.748	1.866
cathedral.pgm (3008 × 2000)	2.120	2.135	1.734	1.922	2.015	2.126	2.697	2.162	2.406
deer.pgm (2641 × 4043)	1.532	1.504	1.407	1.507	1.480	1.537	1.915	1.567	1.588
fireworks.pgm (2352 × 3136)	5.262	4.496	3.279	3.762	3.121	5.279	5.432	4.624	5.742
flower_foveon.pgm (1512 × 2268)	3.938	3.746	2.806	3.149	3.060	3.927	5.611	4.158	4.744
hdr.pgm (2048 × 3072)	3.255	3.161	2.561	2.653	2.716	3.269	4.614	3.213	3.406
spider_web.pgm (2848 × 4256)	4.411	4.209	3.029	3.365	3.234	4.555	6.017	4.317	4.895



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.16	2.448	0.968	0.513	0.9	0.353	0.25	0.507	0.877
JPEG 2000	1.391	6.757	2.729	1.403	3.063	0.984	0.408	0.942	1.276
Lossless JPEG	0.779	3.24	1.252	0.629	1.24	0.792	0.25	0.606	1.06
PNG	1.195	5.778	2.308	2.042	1.387	2.124	1.529	3.568	6.105
JPGE XR	0.172	1.431	0.52	0.268	0.455	0.269	0.17	0.319	0.491
CALIC	2.0977	8.1549	3.2884	1.7042	3.2036	1.7375	0.8454	2.2511	3.0573
AVIF	1.1	7.01	2.75	1.59	2.91	1.96	1.08	1.54	3.7
WebP	0.96	6.76	2.66	1.42	2.74	1.66	0.76	1.33	3.42
FLIF	3.02	9.06	12.83	5.97	13.85	4.43	2.19	4.86	7.7

Figure 7.5: Comparison of encoding times for 8-bit RGB images



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.196	2.903	1.183	0.639	1.082	0.434	0.3	0.609	0.992
JPEG 2000	0.717	6.494	2.656	1.642	3.008	0.765	0.385	0.96	1.253
Lossless JPEG	0.3	1.283	0.387	0.243	0.395	0.691	0.636	0.212	0.463
PNG	0.225	0.638	0.253	0.132	0.209	0.123	0.062	0.124	0.241
JPGE XR	0.157	1.531	0.486	0.353	0.429	0.227	0.184	0.346	0.491
CALIC	1.6673	7.0439	2.3755	1.8103	3.1052	1.6921	0.7362	2.0391	2.1439
AVIF	0.36	1.56	4.67	2.19	5.11	0.6	0.46	0.71	1.4
WebP	0.32	1.54	4.55	2.1	5.05	0.56	0.35	0.59	1.36
FLIF	0.41	1.73	4.75	2.379	5.75	0.36	0.62	0.53	1.64

Figure 7.6: Comparison of decoding times for 8-bit RGB images

DTs and CRs. From Table 7.3, it can be seen that FLIF gives the highest CRs for all the images except deer.pgm, flower_foveon.pgm, and hdr.pgm. For these images, AVIF provides the highest CRs (3.742, 8.273, and 7.779, respectively). Lossless JPEG gives the lowest CR (2.791) for artificial.pgm, and PNG gives the lowest for the rest of the images. Figure 7.8 shows that FLIF gives the highest ETs for all the images except big_tree.pgm. For this image, JPEG 2000 takes the highest (4.579 s). JPEG-LS gives the lowest ETs (0.186, 0.238, 0.257 and 0.406 s) for four images (artificial, cathedral, fireworks, and spider_web, respectively) and JPEG XR gives the lowest for the rest of the images. In terms of decoding, CALIC and lossless JPEG give the highest DTs (0.613 and 0.503 s) for artificial.pgm and flower_foveon.pgm, respectively, and JPEG 2000 gives the highest DTs (1.078, 0.978, and 1.736 s) for three images (fireworks, hdr, and spider_web, respectively). For the rest of the images, FLIF takes the highest DTs. On the otherhand, PNG gives the lowest DTs for all images shown in Figure 7.9.

The 16-bit RGB images and their CRs, ETs and DTs are shown in Figure 7.10, Table 8 and Figures 7.11-7.12, respectively. From Table 7.4, it can be seen that FLIF gives the highest CRs (37.021 and 12.283) for artificial.ppm and fireworks.ppm, respectively and AVIF for the rest of the images. Lossless JPEG gives the lowest CRs (2.6952 and 1.6879) for the artificial and fireworks images, respectively, and PNG gives the lowest for the rest of the images. In terms of the encoding time, JPEG 2000 gives the highest (14.259, 5.699 and 5.672 s) for three images (big_tree.pgm, bridge.pgm, and fireworks.pgm, respectively), and FLIF gives the highest ETs for the rest of the images in Figure 7.11. JPEG XR takes the lowest ETs for all the images. Figure 7.12 shows that CALIC gives the longest DT (2.278 s) for artificial.ppm, and FLIF gives the longest DTs (2.05, 3.27, and 5.01 s) for three images (flower_foveon.pgm, hdr.pgm, and spider_web.pgm, respectively), while JPEG 2000 takes the longest DTs for the rest of the images. PNG gives the shortest DT (0.292, and 0.28s) for fireworks.pgm and hdr.pgm, respectively, and JPEG XR gives the shortest for the rest of the images. It should be noted that although Figures 7.1 and 7.7, and Figures 7.4 and 7.10 are visually identical, these four figures are completely different in terms of the bit depths and channels in the images.

Figure 7.13 shows the average CRs for all four types of image. It can be seen that FLIF gives the highest (4.451, 9.013, 5.002, and 10.114) CRs for all type of images and achieves compression that is 10.99%, 23.19%, 40.1%, 26.2%, 43.14%, 7.73%, 26.38%, and 13.46% higher for the 8-bit greyscale images, 79.97%, 80.37%, 82.56%, 81.98%, 43.65%, 79.68%, 26.72%, and 14.01% higher for the 16-bit greyscale images, 23.43%, 31.09%, 49.18%, 31.97%, 48.02%, 22.75%, 16.41%, and 8.92% higher for the 8-bit RGB images, and 81.51%, 81.83%, 84.76%, 82.91%, 48.34%, 81.32%, 16.84%, and 7.65% higher for the 16-bit RGB images than JPEG-LS, JPEG 2000, Lossless JPEG, PNG, JPEG XR, CALIC, AVIF, and WebP, respectively. If the CR is the main consideration for an application, we can see from Figure 7.13 that FLIF is better for all four types of image.

Figure 7.14 shows the average encoding times. It can be seen that JPEG XR requires the shortest average ET (0.167, 0.376, 0.455 and 0.417 s) for the 8-bits greyscale, 16-bit greyscale, 8-bit RGB and 16-bit RGB images, respectively.In terms of encoding, JPEG XR is on average 32.39%, 72.03%, 48.77%, 85.17%, 82.49%, 92.08%, 87.44%, and 96.73% faster for 8-bit greyscale images, 20.84%, 75.84%, 19.66%, 69.41%, 55.4%, 79.64%, 75.49%, and 85.35% faster for 16-bit greyscale images, 41.29%, 78.4%, 58.41%, 84.27%, 84.46%, 82.68%, 81.14%, and 93.59% faster for 8-bit RGB images, and 61.95%, 91.15%, 72.96%, 88.64% and 83.42%, 84.65%, 83.85%, and 91.63% faster for 16-bit RGB images than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF respectively. If the ET is the main consideration for an application, we can draw the conclusion that compression using JPEG XR is best for all types of images.

The average DT is shown in Figure 7.15. PNG gives the shortest DT (0.072, 0.161 and 0.223 s) for the 8-bit greyscale, 16-bit greyscale and 8-bit RGB images, respectively, while JPEG XR gives the shortest (0.398 s) for the 16-bit RGB images. On average, PNG decodes 74.19%,

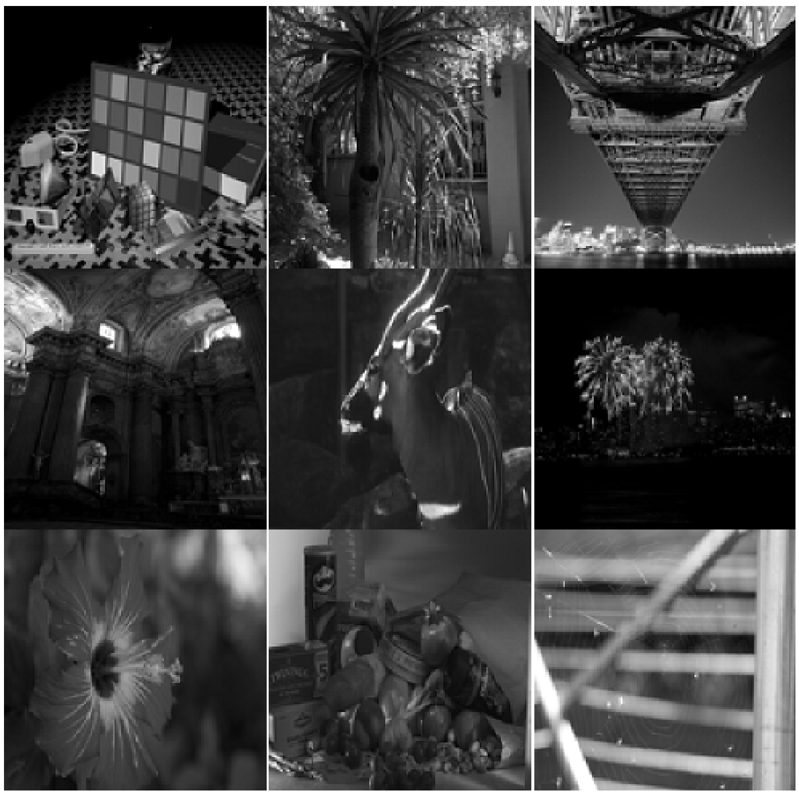
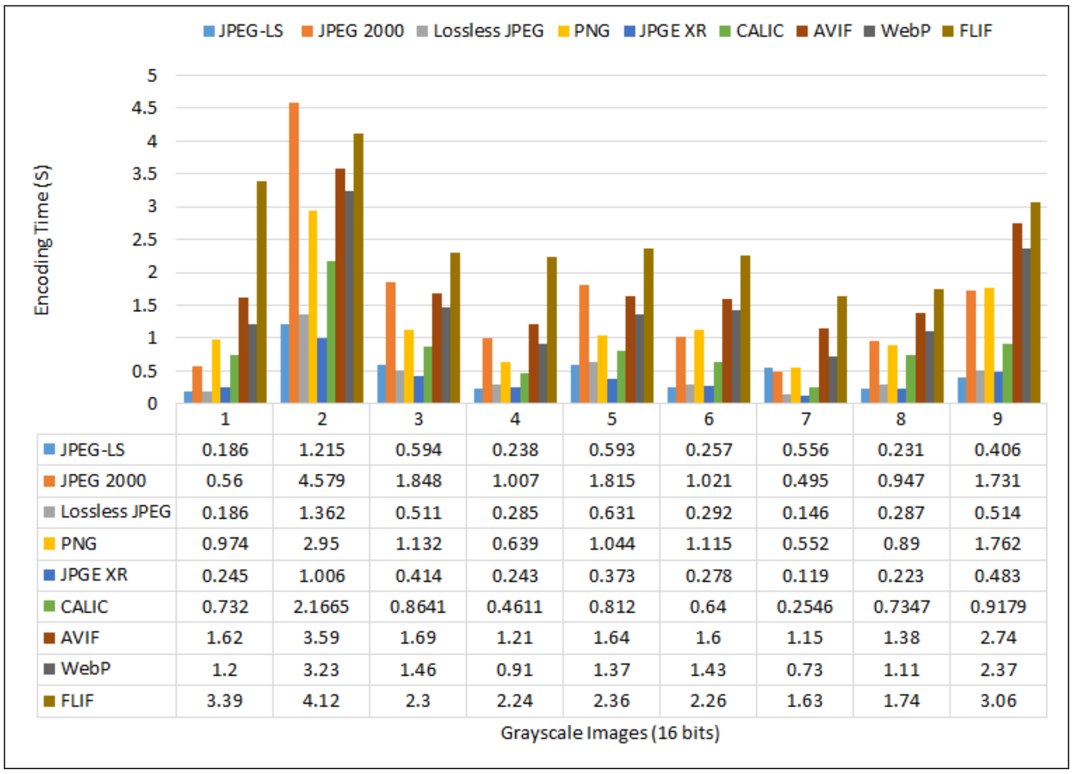
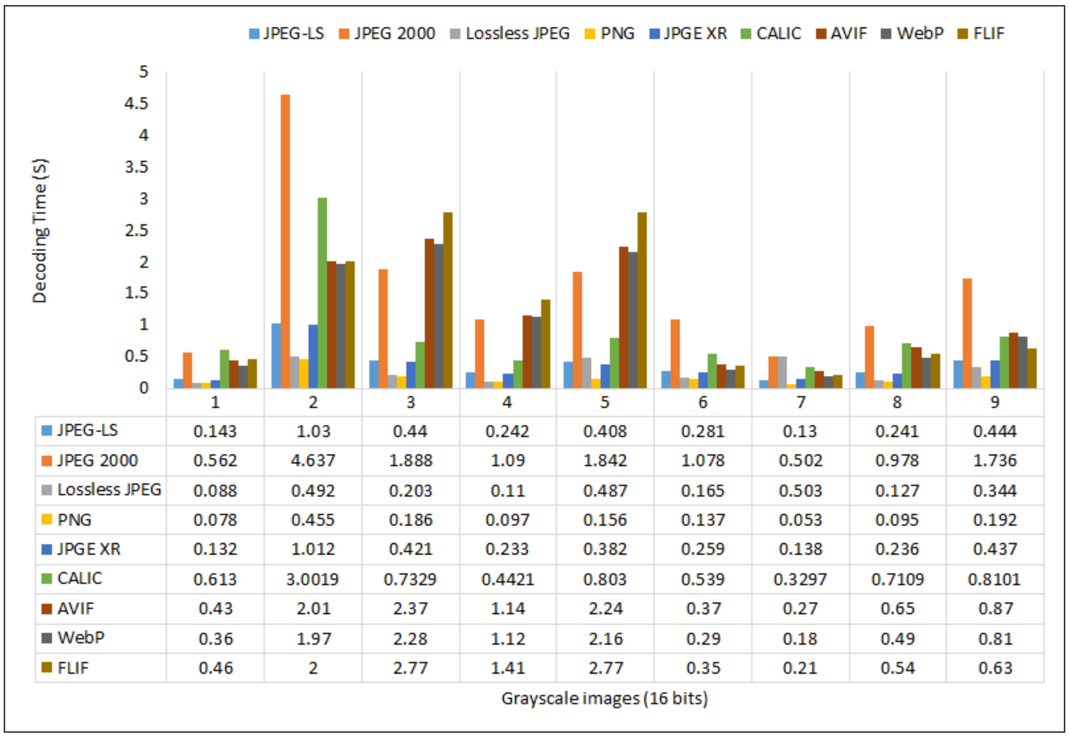


Figure 7.7: Examples of 16-bit greyscale images



	1	2	3	4	5	6	7	8	9
🟦 JPEG-LS	0.186	1.215	0.594	0.238	0.593	0.257	0.556	0.231	0.406
🟧 JPEG 2000	0.56	4.579	1.848	1.007	1.815	1.021	0.495	0.947	1.731
⬜ Lossless JPEG	0.186	1.362	0.511	0.285	0.631	0.292	0.146	0.287	0.514
🟨 PNG	0.974	2.95	1.132	0.639	1.044	1.115	0.552	0.89	1.762
🟦 JPGE XR	0.245	1.006	0.414	0.243	0.373	0.278	0.119	0.223	0.483
🟩 CALIC	0.732	2.1665	0.8641	0.4611	0.812	0.64	0.2546	0.7347	0.9179
🟥 AVIF	1.62	3.59	1.69	1.21	1.64	1.6	1.15	1.38	2.74
⬛ WebP	1.2	3.23	1.46	0.91	1.37	1.43	0.73	1.11	2.37
🟫 FLIF	3.39	4.12	2.3	2.24	2.36	2.26	1.63	1.74	3.06

Figure 7.8: Comparison of encoding times for 16-bit greyscale images



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.143	1.03	0.44	0.242	0.408	0.281	0.13	0.241	0.444
JPEG 2000	0.562	4.637	1.888	1.09	1.842	1.078	0.502	0.978	1.736
Lossless JPEG	0.088	0.492	0.203	0.11	0.487	0.165	0.503	0.127	0.344
PNG	0.078	0.455	0.186	0.097	0.156	0.137	0.053	0.095	0.192
JPGE XR	0.132	1.012	0.421	0.233	0.382	0.259	0.138	0.236	0.437
CALIC	0.613	3.0019	0.7329	0.4421	0.803	0.539	0.3297	0.7109	0.8101
AVIF	0.43	2.01	2.37	1.14	2.24	0.37	0.27	0.65	0.87
WebP	0.36	1.97	2.28	1.12	2.16	0.29	0.18	0.49	0.81
FLIF	0.46	2	2.77	1.41	2.77	0.35	0.21	0.54	0.63

Figure 7.9: Comparison of decoding times for 16-bit greyscale images

Table 7.3: Comparison of compression ratios for 16-bit greyscale images

Image Name (Dimensions)	**JPGE-LS**	**JPEG 2000**	**Lossless JPEG**	**PNG**	**JPEG XR**	**CALIC**	**AVIF**	**WebP**	**FLIF**
artificial.pgm (2048 × 3072)	4.073	4.007	2.791	4.381	6.393	4.174	8.282	20.846	26.677
big_tree.pgm (4550 × 6088)	1.355	1.325	1.287	1.181	4.028	1.324	4.35	4.229	4.537
bridge.pgm (4049 × 2749)	1.309	1.279	1.244	1.147	3.696	1.373	3.944	3.843	3.971
cathedral.pgm (3008 × 2000)	1.373	1.337	1.293	1.191	4.084	1.342	4.450	4.398	4.749
deer.pgm (2641 × 4043)	1.252	1.241	1.225	1.132	3.371	1.339	3.742	3.549	3.557
fireworks.pgm (2352 × 3136)	1.946	1.809	1.740	1.604	6.116	1.955	8.967	9.773	11.647
flower_foveon.pgm (1512 × 2268)	1.610	1.591	1.523	1.316	5.884	1.663	8.273	7.626	8.029
hdr.pgm (2048 × 3072)	1.595	1.563	1.491	1.297	5.755	1.569	7.779	7.040	7.754
spider_web.pgm (2848 × 4256)	1.736	1.771	1.554	1.367	6.386	1.742	9.658	8.442	10.196

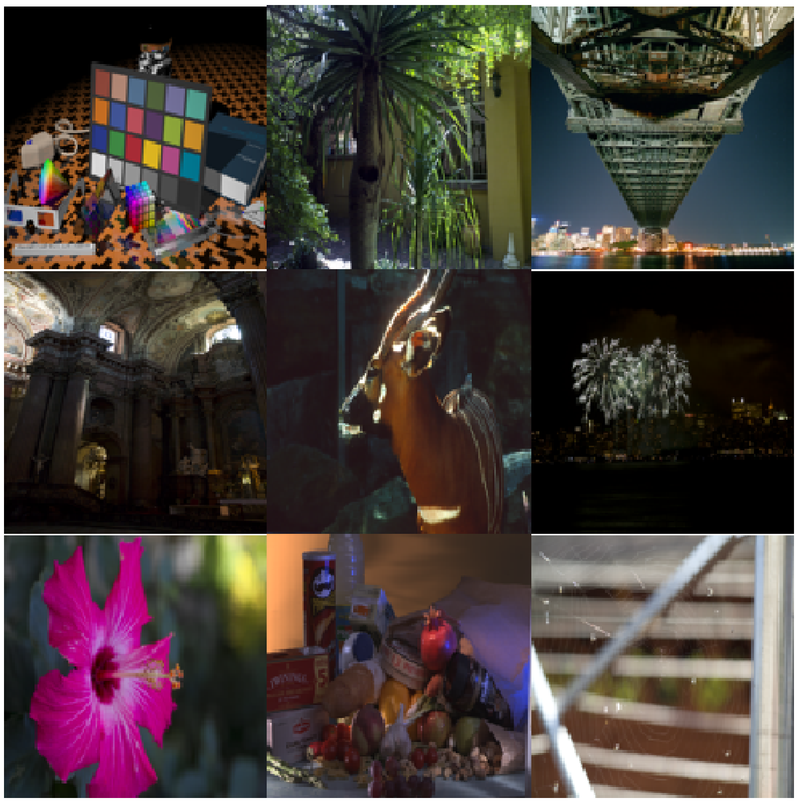
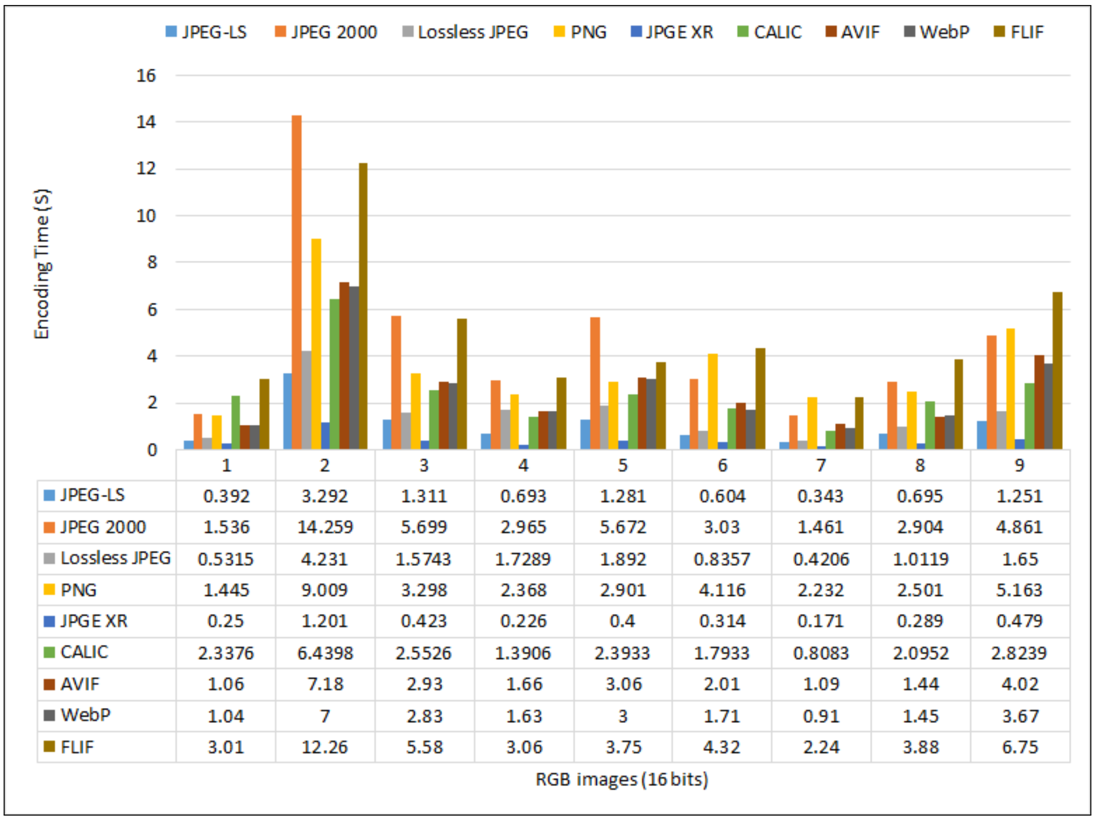
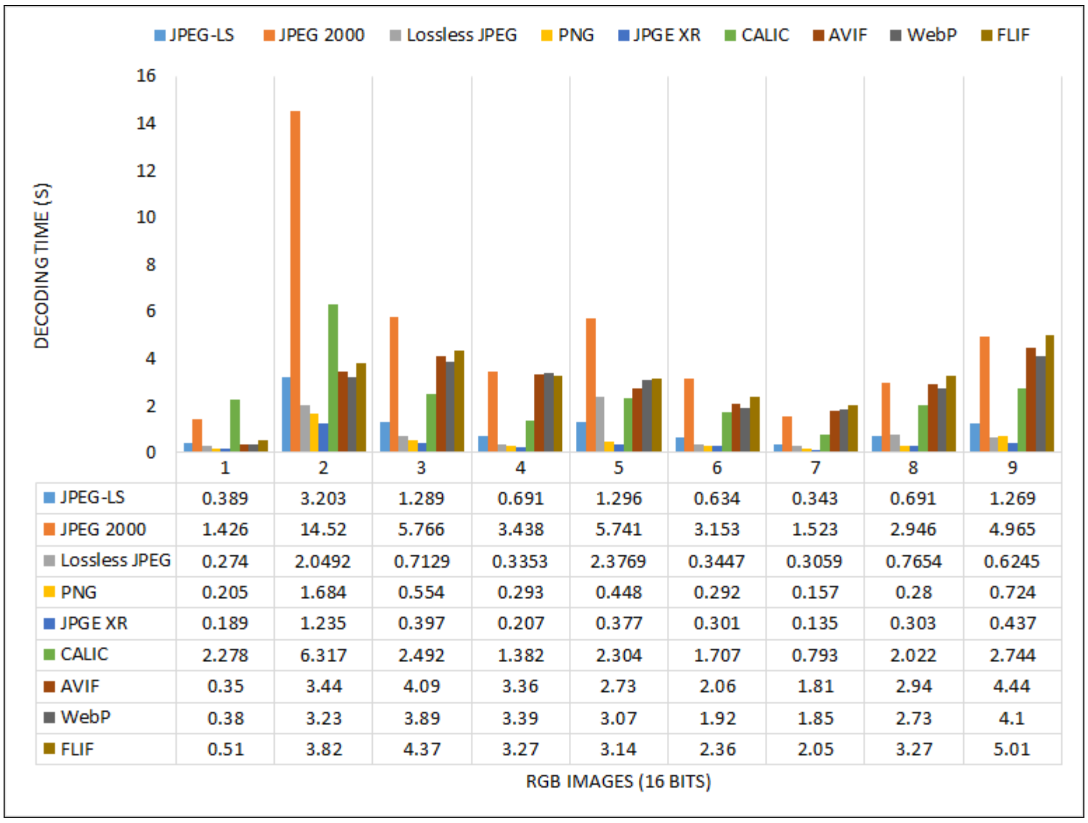


Figure 7.10: Examples of 16-bit RGB images



	1	2	3	4	5	6	7	8	9
🟦 JPEG-LS	0.392	3.292	1.311	0.693	1.281	0.604	0.343	0.695	1.251
🟧 JPEG 2000	1.536	14.259	5.699	2.965	5.672	3.03	1.461	2.904	4.861
⬜ Lossless JPEG	0.5315	4.231	1.5743	1.7289	1.892	0.8357	0.4206	1.0119	1.65
🟨 PNG	1.445	9.009	3.298	2.368	2.901	4.116	2.232	2.501	5.163
🟦 JPGE XR	0.25	1.201	0.423	0.226	0.4	0.314	0.171	0.289	0.479
🟩 CALIC	2.3376	6.4398	2.5526	1.3906	2.3933	1.7933	0.8083	2.0952	2.8239
🟫 AVIF	1.06	7.18	2.93	1.66	3.06	2.01	1.09	1.44	4.02
⬛ WebP	1.04	7	2.83	1.63	3	1.71	0.91	1.45	3.67
🟫 FLIF	3.01	12.26	5.58	3.06	3.75	4.32	2.24	3.88	6.75

Figure 7.11: Comparison of encoding times for 16-bit RGB images



	1	2	3	4	5	6	7	8	9
JPEG-LS	0.389	3.203	1.289	0.691	1.296	0.634	0.343	0.691	1.269
JPEG 2000	1.426	14.52	5.766	3.438	5.741	3.153	1.523	2.946	4.965
Lossless JPEG	0.274	2.0492	0.7129	0.3353	2.3769	0.3447	0.3059	0.7654	0.6245
PNG	0.205	1.684	0.554	0.293	0.448	0.292	0.157	0.28	0.724
JPGE XR	0.189	1.235	0.397	0.207	0.377	0.301	0.135	0.303	0.437
CALIC	2.278	6.317	2.492	1.382	2.304	1.707	0.793	2.022	2.744
AVIF	0.35	3.44	4.09	3.36	2.73	2.06	1.81	2.94	4.44
WebP	0.38	3.23	3.89	3.39	3.07	1.92	1.85	2.73	4.1
FLIF	0.51	3.82	4.37	3.27	3.14	2.36	2.05	3.27	5.01

Figure 7.12: Comparison of decoding times for 16-bit RGB images

Table 7.4: Comparison of compression ratios for 16-bit RGB images

Image Name (Dimensions)	**JPGE-LS**	**JPEG 2000**	**Lossless JPEG**	**PNG**	**JPEG XR**	**CALIC**	**AVIF**	**WebP**	**FLIF**
artificial.pgm (2048 × 3072)	4.335	4.734	2.695	4.896	8.644	4.404	13.783	36.292	37.021
big_tree.pgm (4550 × 6088)	1.302	1.261	1.249	1.150	3.441	1.284	4.504	3.614	3.841
bridge.pgm (4049 × 2749)	1.274	1.243	1.240	1.132	3.470	1.283	4.414	3.445	3.739
cathedral.pgm (3008 × 2000)	1.379	1.333	1.267	1.218	4.036	1.383	5.397	4.292	4.851
deer.pgm (2641 × 4043)	1.197	1.173	1.236	1.100	2.961	1.196	3.826	3.096	3.176
fireworks.pgm (2352 × 3136)	2.378	1.832	1.688	2.053	6.453	2.384	11.318	10.048	12.283
flower_foveon.pgm (1512 × 2268)	1.724	1.664	1.494	1.371	6.121	1.765	11.217	8.305	9.513
hdr.pgm (2048 × 3072)	1.535	1.518	1.473	1.275	5.432	1.586	9.219	6.391	6.819
spider_web.pgm (2848 × 4256)	1.702	1.784	1.531	1.360	6.466	1.718	12.018	8.576	9.787

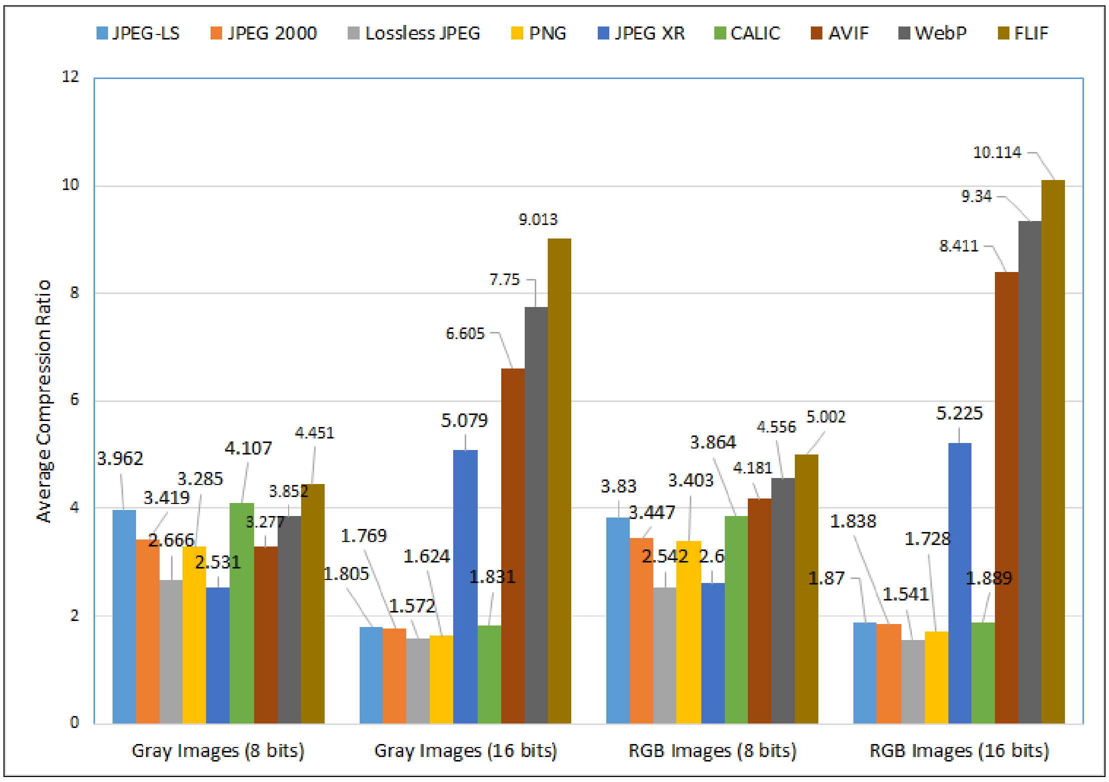


Figure 7.13: Comparison of average compression ratios

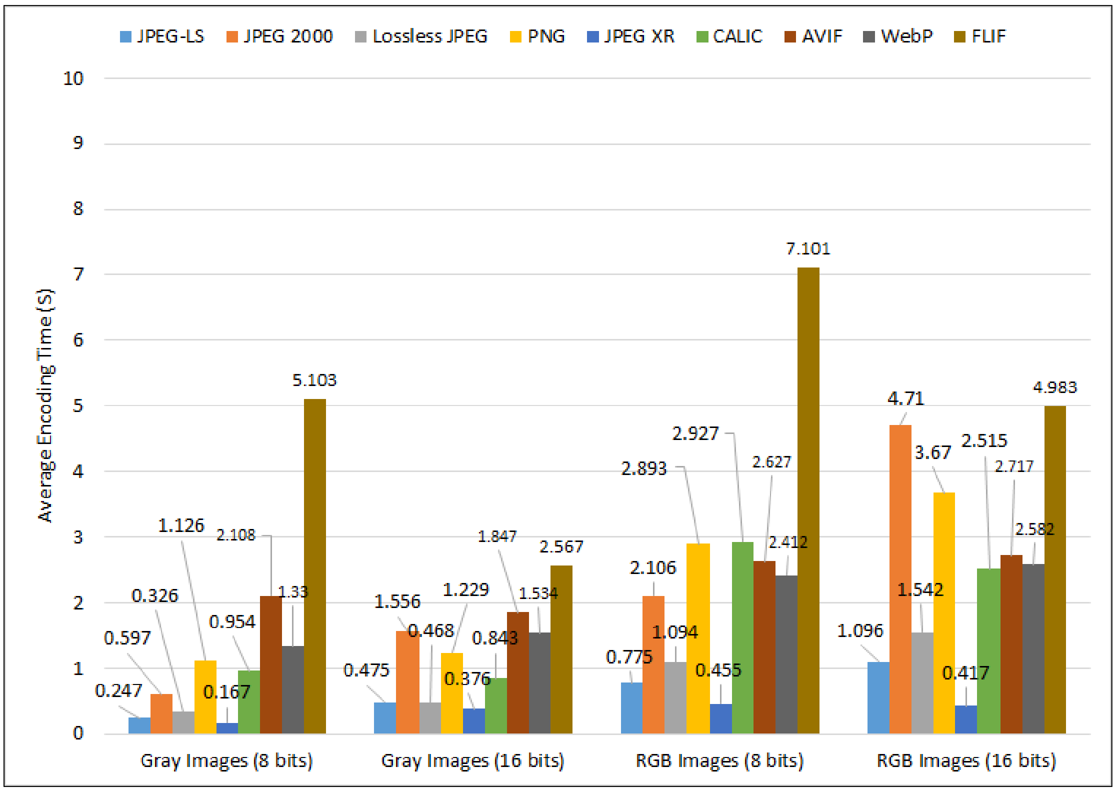


Figure 7.14: Comparison of average encoding times

87.21%, 45.86%, 55.28%, 91.57%, 87.86%, 93.55%, and 93.66% faster for 8-bit greyscale images, 56.84%, 89.87%, 42.5%, 55.4%, 81.85%, 86%, 85% and 87% faster for 16-bit greyscale images, and 75.92%, 88.78%, 56.45%, 52.25%, 91.13%, 88.24%, 87.77% and 88.95% faster for 8-bit RGB images than JPEG-LS, JPEG 2000, lossless JPEG, JPEG XR, CALIC, AVIF, WebP, and FLIF, respectively. However, for 16-bit RGB images, JPEG XR is 63.45%, 91.76%, 53.99%, 22.72%, 83.75%, 85.8%, 85.42% and 87.12% faster than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively, for the same case. When DT is the main consideration for an application during compression, we can conclude from Figure 28 that PNG is better for 8-bit greyscale, 16-bit greyscale and 8-bit RGB images, whereas JPEG XR is better for 16-bit RGB images.

Analysis based on our developed technique

The average performance of each method (MN) for each parameter (PM) is calculated using equation (7.1), where N is the number of images.

$$AV_{(MN,PM)} = \frac{1}{N}\sum_{i=1}^{N} PM, \qquad (7.1)$$
$$where\ PM \in (CR, ET\ \&\ DT)$$

The truth is: "Lossless data compression techniques are mainly evaluated based on compression ratio (CR), encoding time (ET), and decoding time (DT)". Beside, a better lossless image compression is proportional to the compression ratio and inversely proportional to the encoding and decoding times. Since the performance of a lossless algorithm depends on the CR, ET and DT, we develop a technique defined in equations (7.2-7.4) to calculate the overall impact for each algorithm in terms of compression ratio, encoding and decoding times, where K1 is a constant. A better K1 makes a good balance among the compression ratio, encoding and decoding times. The compression ratio makes a good balance with encoding and decoding times if K1 is the average of OVP_{ET} and OVP_{DT}.

$$OVP_{(MN,CR)} = \frac{AV_{(MN,CR)}}{\sum_{All\ methods} AV_{CR}} \times K1 \qquad (7.2)$$

$$OVP_{(MN,ET)} = \frac{\sum_{All\ methods} AV_{ET}}{AV_{(MN,ET)}} \qquad (7.3)$$

$$OVP_{(MN,DT)} = \frac{\sum_{All\ methods} AV_{DT}}{AV_{(MN,DT)}} \qquad (7.4)$$

Considering all of these parameters simultaneously, we calculate the grand total performance of each method using the equation 7.5. Overall performance means how good is each method averagely among all methods in terms of each individual evaluation parameter whereas grand total performance is a combination of two or more parameters.

$$GTP_{(MN)} = \frac{\sum_{k=1}^{NP} OVP_{MN}}{\sum_{All\ methods}\sum_{k=1}^{NP} OVP_{MN}} \times 100, \qquad (7.5)$$
$$where\ 2 \leq NP \leq TP)$$

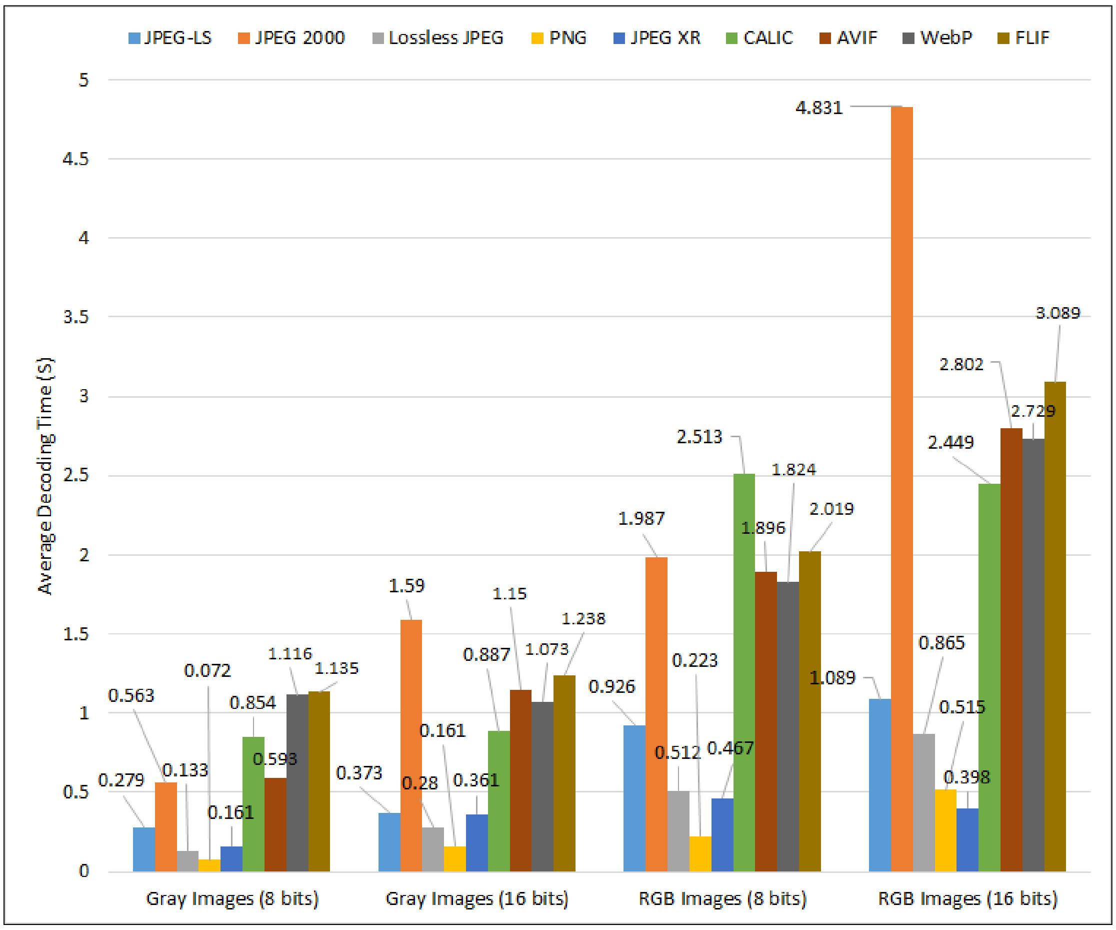


Figure 7.15: Comparison of average decoding times

In the equation 7.5, NP is the number of parameters considered when calculating the grand total, and TP is the total number of parameters. For lossless image compression, TP can be a maximum of three because only three parameters (compression ratio, encoding and decoding times) are used for evaluating a method. Finally, we apply the Algorithm 7 to find a better lossless image compression algorithm.

Algorithm 7: Selected Method Name (SMN)

```
1  BestTechnique(NP, MethodList) ;
2    Initialize Max ← 0;
3    SM ← 0;
4    K ← 1;
5    NM ← Number of methods;
6    P ← GTP(NP,NM);
7    while K≤ NM do
8      if Max<P then
9        Max ← P;
10       SM ← K;
11     end
12 end
13   Return MethodList[SM];
```

We show the two-parameter GTP (i.e. the GTP for each combination of two parameters) in Figures 7.16–7.19 for 8-bit greyscale, 8-bit RGB, 16-bit greyscale, and 16-bit RGB images, respectively. Figure 7.16(a) shows that JPEG XR and AVIF give the highest (25.128%) and the lowest (5.068%) GTPs, respectively, in terms of CR and ET. Figure 7.16(b) shows that PNG and WebP provide the highest (27.607%) and the lowest (5.836%) GTPs, respectively, in terms of CR and DT. For ET and DT, Figure 7.16(c) shows that JPEG XR and FLIF provide the highest (25.429%) and the lowest (1.661%) GTPs, respectively. We can draw the conclusion from Figure 7.16 that JPEG XR is better when CR and ET are the main considerations for an application, and this method performs 23.43%, 61.23%, 43.32%, 73.59%, 67.91%, 79.83%, 73.38%, and 79.34% better than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP and FLIF, respectively. PNG is better when CR and DT are the main considerations for an application, and this method performs 61.63%, 75.11%, 42.28%, 50.99%, 76.11%, 76.25%, 78.86%, and 76.54% better than JPEG-LS, JPEG 2000, Lossless JPEG, JPEG XR, CALIC, AVIF, WebP, and FLIF, respectively. JPEG XR is better when ET and DT are the main considerations for an application, and performs 35.35%, 71.84%, 27.93%, 22.84%, 82.09%, 86.34%, 86.89%, and 93.47% better than JPGE-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively.

Figure 7.17 shows the two-parameter GTP for 8-bit RGB images. Figure 7.17(a) shows that JPEG XR and PNG give the highest (23.238%) and the lowest (7.337%) GTP of the state-of-the-art methods in terms of CR and ET, and JPEG XR performs 29.1%, 63.13%, 50.68%, 68.43%, 67.79%, 62.94%, 59.62%, and 68% better than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC AVIF, WebP, and FLIF, respectively. In terms of CR and DT, PNG and CALIC achieve the highest (26.291%) and lowest (6.260%) GTPs, as shown in Figure 7.17(b), and PNG performs 62.09%, 74.69%, 51.58%, 47.77%, 76.19%, 70.87%, 68.75%, and 67.7% better than JPEG-LS, JPEG 2000, lossless JPEG, JPEG XR, CALIC, AVIF, WebP, and FLIF, respectively. Figure 30(c) shows the GTP based on ET and DT. It can be seen that JPEG XR and FLIF have the highest (25.609%) and lowest (3.139%) GTPs. JPEG XR performs 44.19%, 77.73%, 41.05%, 16.51%, 83.39%, 80.12%, 78.78%, and 66.84% better than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively. Finally, we can conclude from Figure 30 that JPEG XR is better for 8-bit RGB image compression when either CR and

ET or ET and DT are the major considerations for an application. However, PNG is better when CR and DT are most important.

For 16-bit greyscale images, Figure 7.18 shows the two-parameter GTP for the state-of-the-art techniques. In terms of CR and ET, JPEG XR achieves the highest GTP (19.305%) and JPEG 2000 the lowest (5.327%) of the methods shown in Figure 7.18(a). It can also be seen that JPEG XR performs 34.87%, 72.41%, 35.54%, 68.96%, 58.53%, 44.38%, 34.31%, and 33.01% better than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively. Figure 7.18(b) shows the GTP based on CR and DT, and it can be seen that PNG and JPEG 2000 have the highest (20.288%) and lowest (3.866%) GTPs. Figure 7.18(b) shows that PNG performs 50.69%, 80.94%, 38.95%, 31.16%, 73.63%, 50.5%, 43.21%, and 38% better than JPEG-LS, JPEG 2000, lossless JPEG, JPEG XR, CALIC, AVIF, WebP, and FLIF, respectively. PNG and FLIF achieve the highest (20.352%) and lowest (3.833%) GTPs in terms of ET and DT, as shown in Figure 7.18(c), and PNG performs 20.81%, 78.37%, 8.23%, 8.23%, 60.52%, 77.22%, 74.12%, and 81.17% better than JPEG-LS, JPEG 2000, lossless JPEG, JPEG XR, CALIC, AVIF, WebP, and FLIF, respectively. Hence, based on Figure 7.18, we conclude that JPEG XR is better for 16-bit greyscale image compression when either CR and ET is the main considerations for an application. PNG is better when CR and DT or ET and DT are considered important for this type of image.

Figure 7.19 shows the GTP of each method for 16-bit RGB images. Figures 7.19(a-c) show that JPEG XR achieves the highest GTP (29.348%, 24.075%, and 35.779%), and JPEG 2000 the lowest (3.963%, 3.340%, and 3.069%) in all cases. JPEG XR performs 62.35%, 86.5%, 72.52%, 84.8%, 80.12%, 58.93%, 55.12%, and 58.91% better in terms of CR and ET; 63.61%, 86.13%, 57.44%, 31.96%, 79.78%, 55.1%, 51.08%, and 59.52% better in terms of CR and DT; and 62.62%, 91.42%, 64.46%, 59.11%, 83.57%, 85.16%, 84.55%, and 89.61%, better in terms of ET and DT than JPEG-LS, JPEG 2000, lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively. We can conclude from Figure 7.19 that JPEG XR is best for 16-bit RGB image compression for any application.

Figure 7.20 shows the three-parameter GTP (CR, ET and DT) for the state-of-the-art techniques. Figures 7.20 (a, b, c, and d) show that JPEG XR achieves the highest GTP (21.957%, 21.120%, 17.306% and 29.865%) for 8-bit greyscale, 8-bit RGB, 16-bit greyscale and 16-bit RGB images, respectively. JPEG XR performs 28.65%, 64.05%, 25.5%, 19.01%, 71.57%, 77.9%, 76.75%, and 81.13% better for 8-bit greyscale images, 35.59%, 67.36%, 37.4%, 12.05%, 72.11%, 66.88%, 64.3%, and 70.81%, better for 8-bit RGB images, 24.88%, 73.95%, 15.19%, 7.97%, 58.77%, 52.03%, 44.43%, and 44.95% better for 16-bit greyscale images, 62.79%, 88.61%, 65.1%, 59.94%, 81.57%, 69.69%, 67.26%, and 70.21% better for 16-bit RGB images than JPEG-LS, JPEG 2000, Lossless JPEG, PNG, CALIC, AVIF, WebP, and FLIF, respectively. We can therefore conclude from Figure 7.20 that JPEG XR is better for the four types of image when all three parameters (CR, ET and DT) are equally important for lossless still image compression for an application. To allow for a straightforward comparison, we summarise the analysis presented in this paper in Table 7.5, and the two best methods are shown in each category.

In this research work, Matlab (version 9.8.0.1323502 (R2020a)) was used for these experiments, on a DELL desktop computer with an Intel(R) Core(TM) i7-8700 CPU @3.20GHz 3.19GHz (Intel, Santa Clara, CA, USA).

7.3 Summary

Lossless still image compression is a topic of intense research interest in the field of computing, especially for applications that rely on low bandwidth connections and limited storage. There are various types of images, and different algorithms are used to compress these in a lossless way. The performance of a lossless algorithm depends on the CR, ET and DT, and there

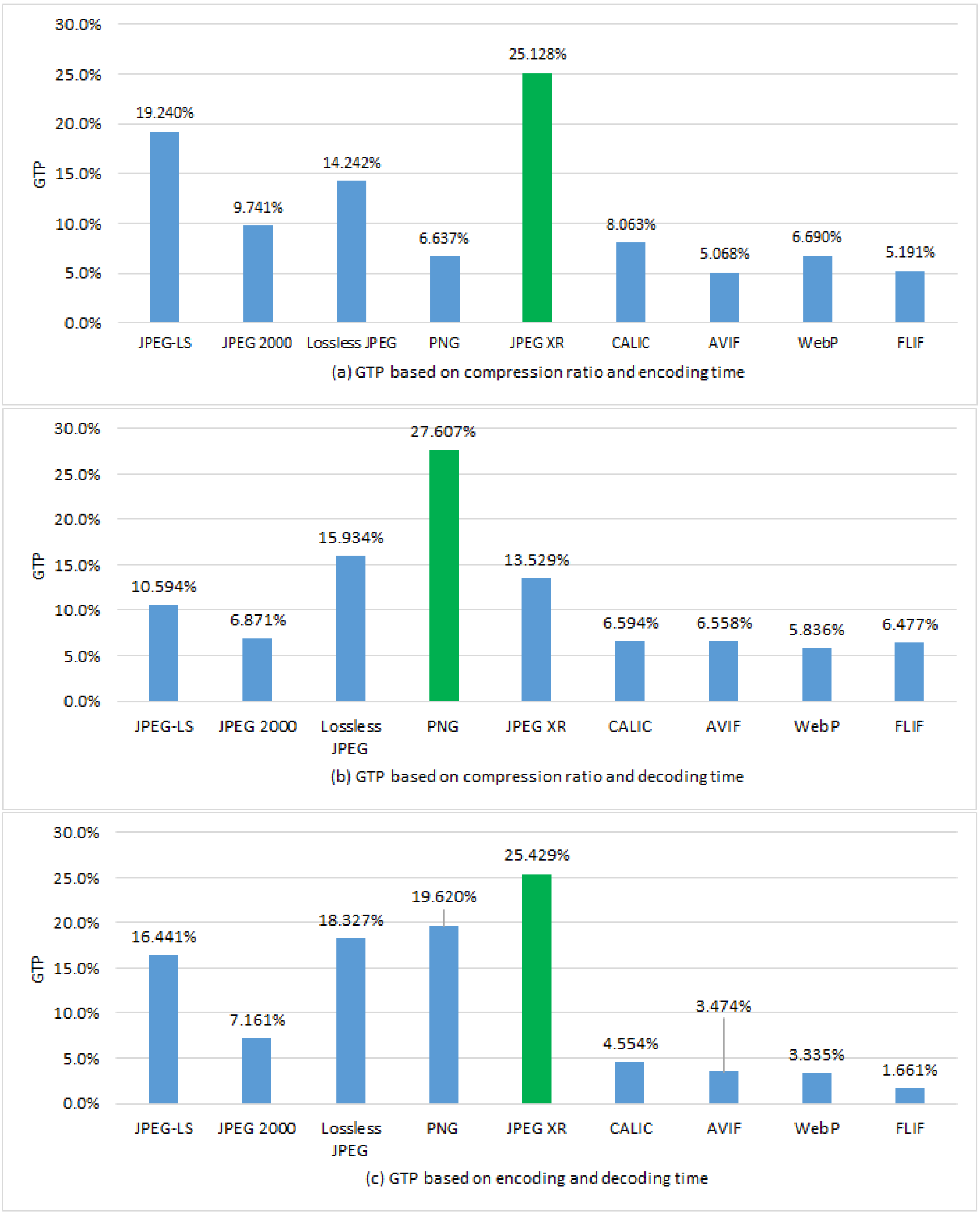


Figure 7.16: Two-parameter GTP for 8-bit greyscale images

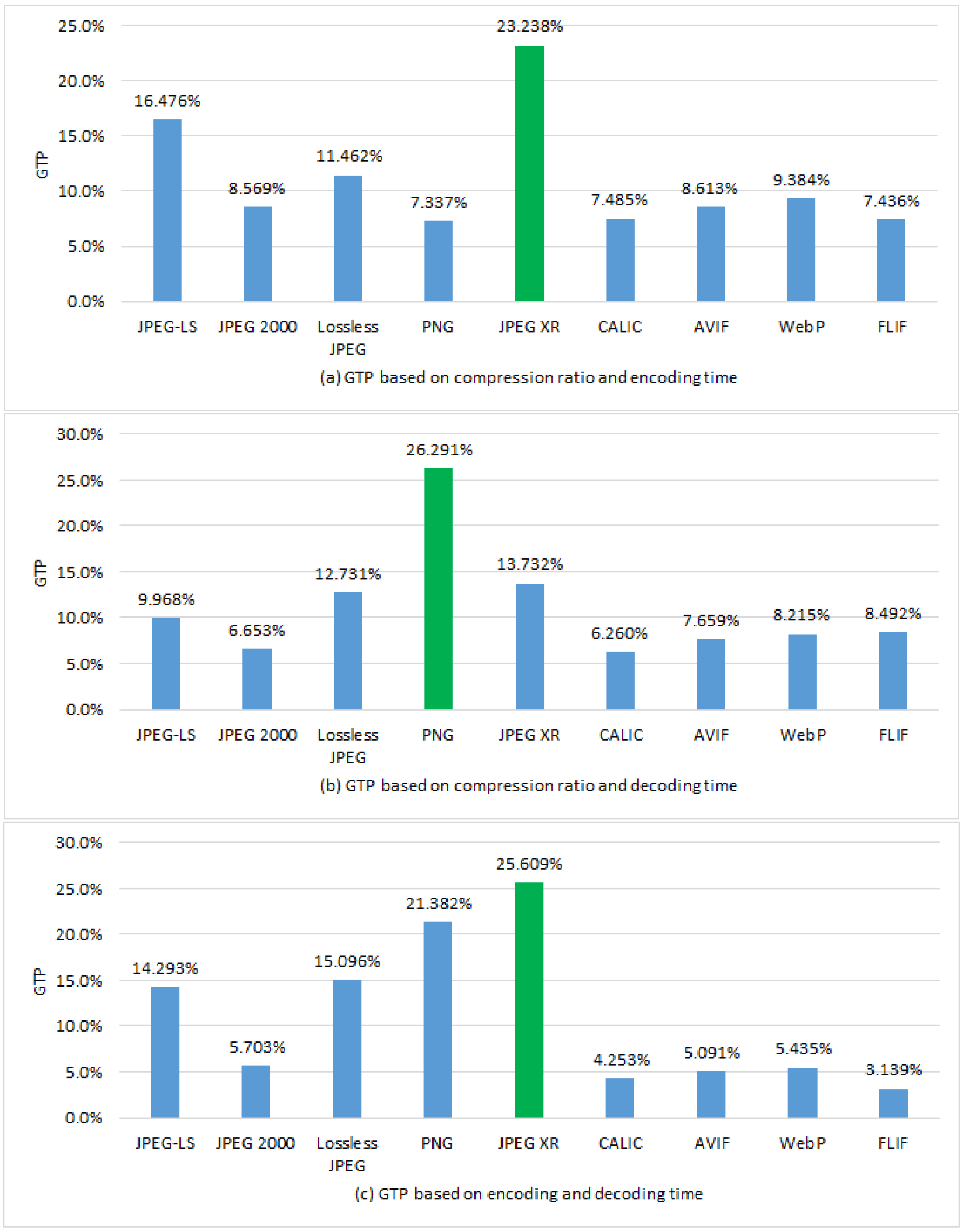


Figure 7.17: Two-parameter GTP for 8-bit RGB images

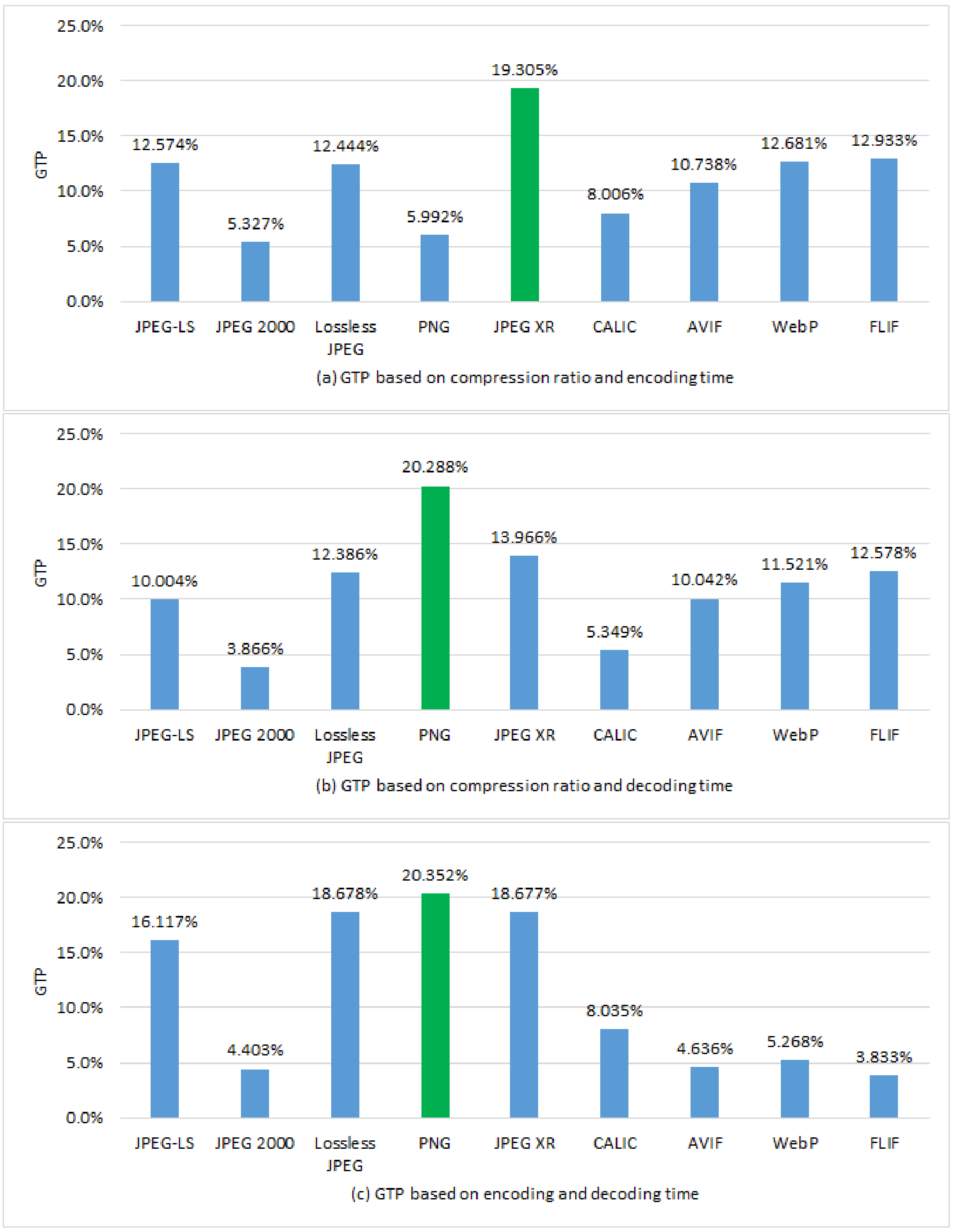


Figure 7.18: Two-parameter GTP for 16-bit greyscale images

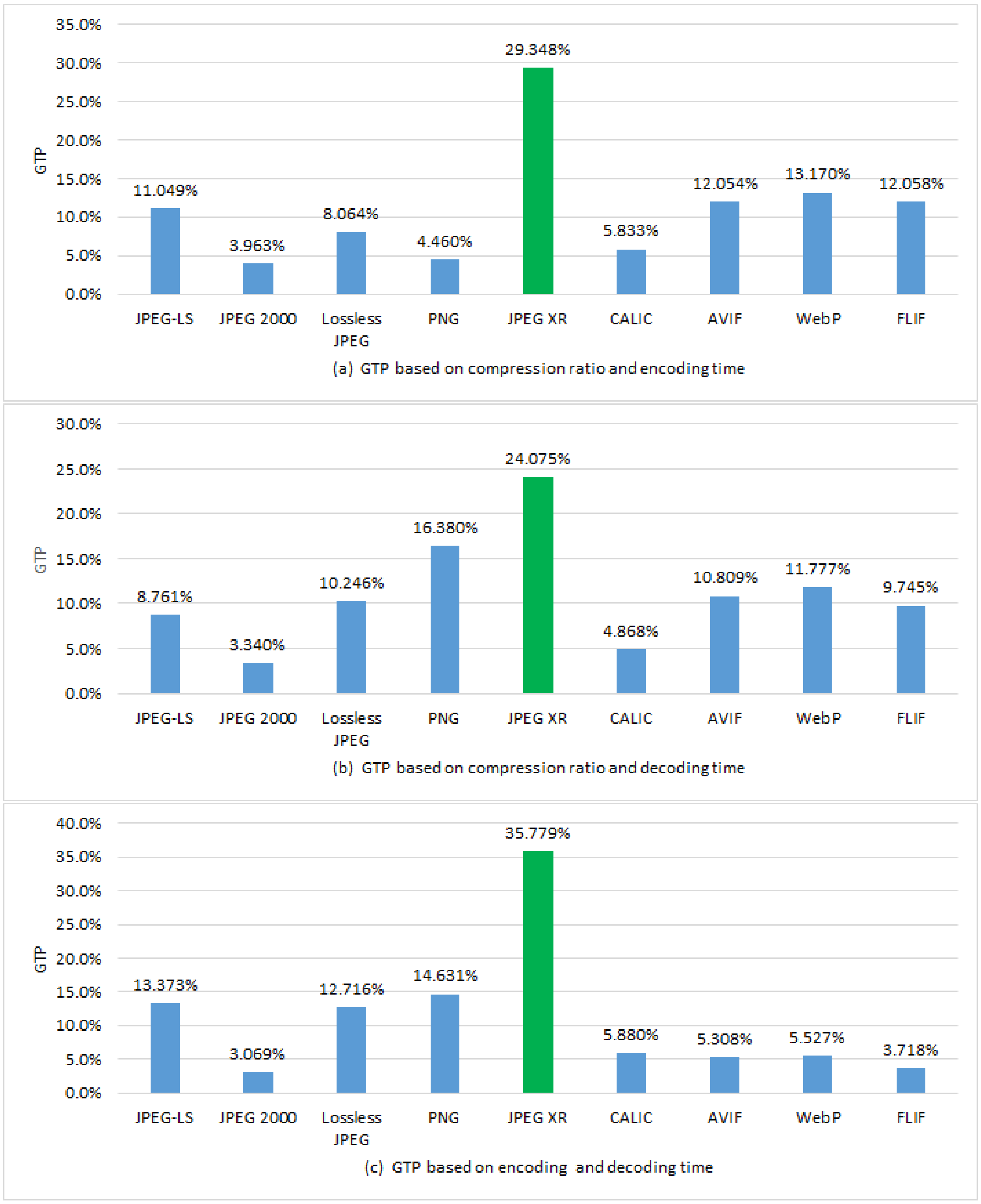
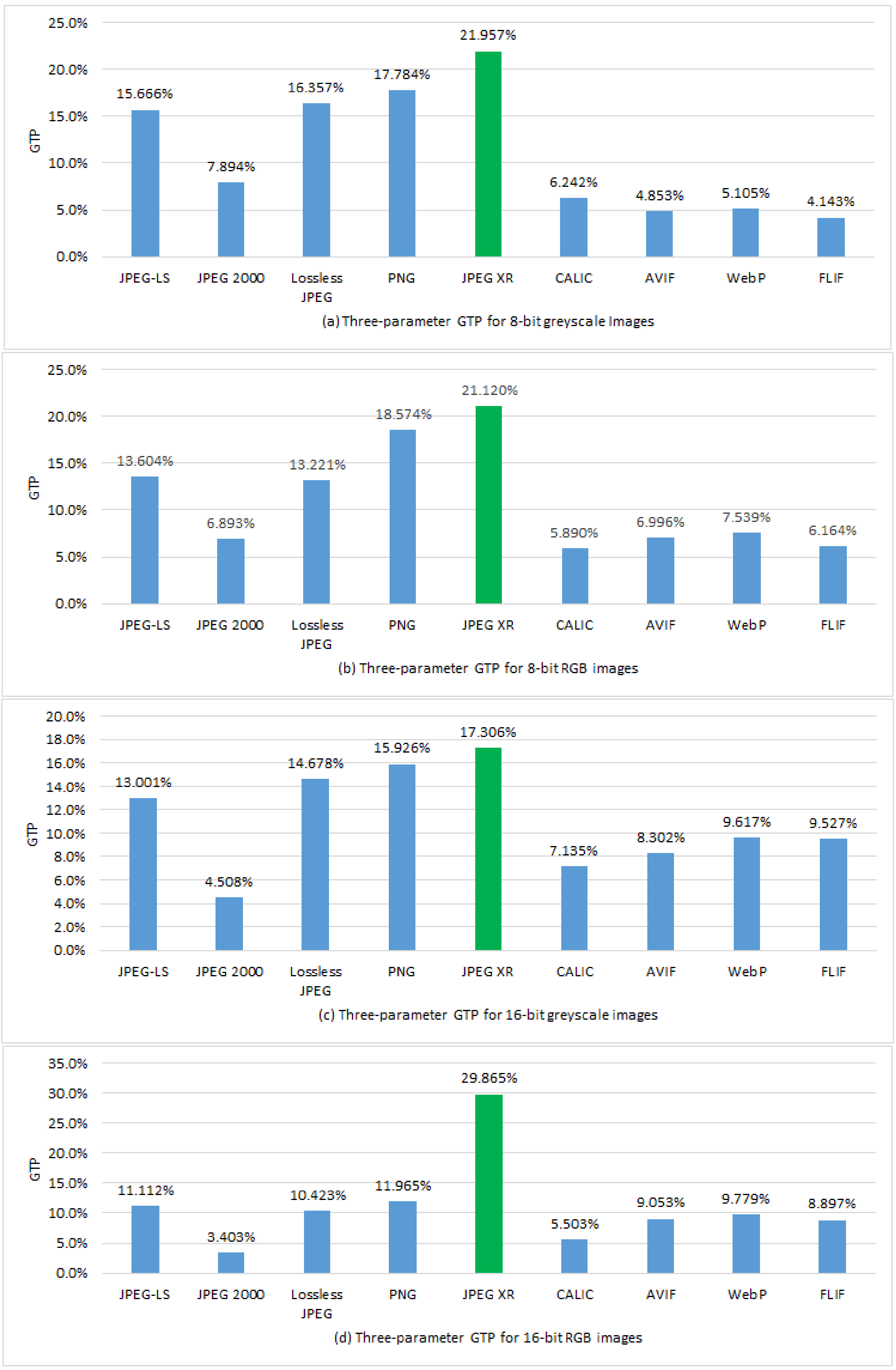


Figure 7.19: Two-parameter GTP for 16-bit RGB images



(a) Three-parameter GTP for 8-bit greyscale Images

(b) Three-parameter GTP for 8-bit RGB images

(c) Three-parameter GTP for 16-bit greyscale images

(d) Three-parameter GTP for 16-bit RGB images

Figure 7.20: Three-parameter GTP

Table 7.5: Summary of results

Grayscale Images (8 bits)				
Compression ratio	**Encoding time**	**Decoding Time**	**First choice**	**Second choice**
✓	✓	✓	JPEG XR	PNG
✓	✓	-	JPEG XR	JPEG-LS
✓	-	✓	PNG	Lossless JPEG
-	✓	✓	JPEG XR	PNG
✓	-	-	FLIF	CALIC
-	-	✓	PNG	Lossless JPEG
-	✓	-	JPEG XR	JPEG-LS
Grayscale Images (16 bits)				
✓	✓	✓	JPEG XR	PNG
✓	✓	-	JPEG XR	FLIF
✓	-	✓	PNG	JPEG XR
-	✓	✓	PNG	Lossless JPEG
✓	-	-	FLIF	WebP
-	-	✓	PNG	Lossless JPEG
-	✓	-	JPEG XR	Lossless JPEG
RGB Images (8 bits)				
✓	✓	✓	JPEG XR	PNG
✓	✓	-	JPEG XR	JPGE-LS
✓	-	✓	PNG	JPGE-XR
-	✓	✓	JPEG XR	PNG
✓	-	-	FLIF	WebP
-	-	✓	PNG	JPEG XR
-	✓	-	JPEG XR	JPEG-LS
RGB Images (16 bits)				
✓	✓	✓	JPEG XR	PNG
✓	✓	-	JPEG XR	WebP
✓	-	✓	JPEG XR	PNG
-	✓	✓	JPEG XR	PNG
✓	-	-	FLIF	WebP
-	-	✓	JPEG XR	PNG
-	✓	-	JPEG XR	JPEG-LS

is a range of demand in terms of different types of performance. Some applications place more importance on the CR, and others focus mainly on the ET or DT. Two or all three of these parameters may be equally important in some applications. The main contribution of this research article is that we have analysed state-of-the-art techniques from each perspective, and have evaluated the strengths of each algorithm for each kind of image. We also recommend the best two state-of-the-art methods from each standpoint.

From the analysis presented here, we can see that FLIF is optimal for the four types of images, in terms of the CR. However, JPEG XR and PNG provide better performance in terms of encoding and decoding speeds, respectively, for 8-bit greyscale and RGB, and 16-bit greyscale. For 16-bit RGB images, JPEG XR works fastest. When the CR and ET are the main considerations, JPEG XR provides better performance for the four types of image. PNG achieves good performance for 16-bit greyscale images when encoding and decoding times are most important, and JPEG XR performs best for other types of images. When CR and DT are most important, PNG is better for 16-bit greyscale, 8-bit greyscale and RGB images, and JPGE-XR is better for 16-bit RGB images. If all parameters are equally important for lossless compression in an application, JPEG XR are better for the four types of image.

An important outcome of this research is that it can allow users to easily identify the optimal compression algorithm to use for an application, based on their particular needs. For example, there are many data storage applications like Google Drive, OneDrive, Dropbox etc., where compression ratio is more important. In this case, FLIF could be the best choice for all types of image. On the other hand, compression ratio and encoding time are more important than decoding time during photo attachment in a mail, and in instant messaging when photo is shared, all the three parameters are equally important. For this two cases, JPEG XR could be the best selection for all categories of image.

Chapter 8

Burrows–Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding

Text compression is one of the most significant research fields, and various algorithms for text compression have already been developed. This is a significant issue, as the use of internet bandwidth is considerably increasing. This article proposes a Burrows–Wheeler transform and pattern matching-based lossless text compression algorithm that uses Huffman coding in order to achieve an excellent compression ratio. In this article, we introduce an algorithm with two keys that are used in order to reduce more frequently repeated characters after the Burrows–Wheeler transform. We then find patterns of a certain length from the reduced text and apply Huffman encoding. We compare our proposed technique with state-of-the-art text compression algorithms. Finally, we conclude that the proposed technique demonstrates a gain in compression ratio when compared to other compression techniques.

8.1 Introduction

Managing the increasing amount of data that are produced by modern daily life activities is not a simple task. In articles [212,213], it is reported that, on average, 4.4 zettabytes and 2.5 exabytes of data were produced per day in 2013 and 2015, respectively. On the other hand, the use of the internet is increasing. The total numbers of internet users were 2.4, 3.4, and 4.4 billion in 2014, 2016, and 2019, respectively [214]. Though hardware manufacturing, companies are producing plenty of hardware in an attempt to provide a better solution for working with huge amounts of data, it's almost impossible to maintain this data without compression.

Compression is the representation of data in a reduced form [215], so that data can be saved while using a small amount of storage and sent with a limited bandwidth [47, 141, 216, 217]. There are two types of compression techniques: lossless and lossy [218, 219]. Lossless compression reproduces data perfectly from its encoded bit stream, and, in lossy compression, less significant information is removed [152, 220].

There are various types of lossless text compression techniques, such as the Burrows–Wheeler transform (BWT), run-length coding, Huffman coding, arithmetic coding, LZ77, Deflate, LZW, Gzip, Bzip2, Brotli, etc. [221,222]. Some statistical methods assign a shorter binary code of variable length to the most frequently repeated characters. Huffman and arithmetic coding are two examples of this type of statistical method. However, Huffman coding is one of the best algorithms in this category [27]. Some dictionary-based methods, such as LZ77, LZW, etc., create a dictionary of substrings and assign them a particular pointer based on the substring's frequency. In [223], Robbi et al. propose a Blowfish encryption and LZW-based text compression procedure and show a better result than LZW coding. Deflate provides a slightly poor compression, but its encoding and decoding speeds are fast [224]. Although researchers have developed many lossless text compression algorithms, they do not fulfill the current demand; researchers are still trying to develop a more efficient algorithm.

From this point of view, we propose a lossless text compression procedure while using the Burrows–Wheeler transformation and Huffman coding in this paper. In our proposed method, we apply a technique using two keys that reduces only the characters repeated consecutively more than two times after the Burrows-Wheeler transformation. Finally, we find all the patterns that have more frequencies and then apply Huffman coding for encoding. We explain our proposed method in detail and compare it against some popular text compression methods. In this paper, previous work is shown in Section 8.2. The proposed techniques are explained in Section 8.3. In Section 8.4, we present the experimental results and analysis, and we give further research directions. Finally, we conclude the article in Section 8.5.

8.2 Previous Works

Run-length coding is one of the text compression algorithms. It calculates symbols and their counts. When run-length coding is directly applied to data for compression, it sometimes takes more storage than the original data [69]. Shannon–Fano coding generates prefix codes of variable length based on probabilities and provides better results than run-length coding, but it is not optimal, as it cannot produce the same tree in encoding and decoding. David Huffman developed a data compression algorithm, reported in [163], which is normally used as a part of many compression techniques. In this technique, a binary tree is generated by connecting the two lowest probabilities at a time when the root of the tree contains the summation of the two probabilities. The tree is then used to encode each symbol without ambiguity. However, Huffman coding cannot achieve an optimal code length when it is applied directly. Arithmetic coding outperforms Huffman coding in terms of average code length, but it takes a huge amount of time for encoding and decoding.

LZW is a dictionary-based lossless text compression algorithm and an updated version of LZ78 [225]. In this technique, a dictionary is created and initialized with strings all of length one. Subsequently, the longest string in the dictionary that matches the current input data is found. Although LZW is a good text compression technique, it is more complicated due to its searching complexity [45]. Deflate is also a lossless compression algorithm that compresses a text by using LZSS and Huffman coding together, where LZSS is a derivative of LZ77. The Deflate procedure finds all of the duplicate substrings from a text. Subsequently, all of the substrings are replaced by the pointer of the substring that occurred first. The main limitation of Deflate is that the longer and duplicate substring searching is a very lazy mechanism [226]. Gzip is another lossless, Deflate-based text compression algorithm that compresses a text while using LZ77 and Huffman coding [227]. The pseudo-code of LZ77 is reported in the reference [228].

The Lempel–Ziv–Markov chain algorithm (LZMA) that was developed by Igor Pavlov is a dictionary-based text compression technique that is similar to LZ77. LZMA uses a comparatively small amount of memory for decoding and it is very good for embedded applications [229]. Bzip2, on the other hand, compresses only a single file using the Burrows-Wheeler transform, the move-to-front (MTF) transform and Huffman entropy coding techniques. Although bzip2 compresses more effectively than LZW, it is slower than Deflate but faster than LZMA [230]. PAQ8n is a lossless text compression method that incorporates the JPEG model into paq81. The main limitation of PAQ8n is that it is very slow [231, 232]. Brotli, which was developed by Google, is a lossless text compression method that performs compression using the lossless mode of LZ77, a Huffman entropy coding technique and second order context modeling [233]. Brotli utilizes a predefined static dictionary holding 13,504 common words [234,235]. It cannot compress large files well because of its limited sliding window [236].

Burrows–Wheeler transform (BWT) in [237] transforms a set of characters into runs of identical characters. It is completely reversible, and no extra information is stored without the position of the last character. The transformed character set can be easily compressed by run-length coding. The pseudo-codes of the forward and inverse Burrows–Wheeler transforms are reported in [238].

8.3 Propose Method

There are many algorithms used to compress a text. Some examples are Huffman, run-length, LZW, Bzip2, Deflate, Gzip, etc. coding-based algorithms [239–243]. Many algorithms focus on the encoding or decoding speed during text compression, while others concentrate on the average code length. Brotli provides better compression ratios than other state-of-the-art techniques for text compression. However, it uses a large static dictionary [235]. What makes our proposal special? We can apply our proposed method to a large file as well as a small file. The main limitation of BWT is that it takes huge amounts of storage and a lot of time to transform a large file [244–246]. Our first interesting innovation is that we split a large file into a set of smaller files, where each file contains the same number of characters, and then apply the Burrows–Wheeler transform (BWT) to the smaller files individually to speed up the transformation. We do not use any static dictionary because searching for a word or phrase in a dictionary is very complicated and time consuming [27]. We change the use of run-length coding a bit after the Burrows–Wheeler transform (BWT), because run-length coding takes the symbol and its count, and it only works well when characters are repeated more in a text. When a character is alone in a text, which normally happens, two values (the character and its count) are stored after encoding, which increases the use of storage. Our second interesting change is that we will only replace the characters repeated more than three times in a row by a key, the character, and its count. The position of the character's sequence in a reduced text is identified by the key. Huffman coding provides more compression if a text has a higher frequency of characters. We have analyzed ten large text files from [30], and the outcomes are shown in Figure 8.1. The figure shows that the frequency of lowercase letters in any text is always much higher than other types of letters.

Figure 8.1 shows that, on average, all of the files contain 71.46%, 3.56%, 15.48%, 2.40%, 1.14%, and 5.96% small letters, capital letters, space, newline, zero to nine, and others, respectively. We have calculated that, averagely, the frequency of lowercase letters is 94.95%, 78.11%, 96.67%, 98.31%, and 91.72% higher than the frequency of capital letters, spaces, newlines, zero-to-nine (0–9) characters, and other characters, respectively.

Additionally, we have analyzed 5575 small text files of lengths less than or equal to 900 characters. We see that a maximum of twenty of the same characters are repeated consecutively at a time after the Burrows–Wheeler transform is applied to the small texts that are shown in Figure 8.2. There are twenty-six lowercase characters in the English alphabet. Accordingly, we have replaced the character count in lowercase letters using the formula (character's count + 96), so that the lowercase letters keep the frequency higher and we can obtain a higher compression ratio. The proposed above-mentioned idea can only reduce the characters repeated four times or more at a time. However, a file can contain many other characters that can be repeated two or three times. Our third interest is to reduce the characters that are repeated exactly three times using the formula (second key, the character). As a result, we can only store two characters instead of three and further reduce the length of the file. Changing characters that appear twice does not help to reduce the file length, so we keep these characters the same.

We have analyzed eleven large text files after applying the Burrows–Wheeler Transform and the text reduction techniques using the two keys explained above in order to find specific patterns. We find patterns of lengths two, three, four, five and six, and the outcome of the analysis is demonstrated in Figure 8.3. This figure shows that the patterns of length two provide

63.03%, 79.02%, 81.49%, and 83.23% higher frequencies than the other patterns of lengths three, four, five, and six, respectively. This is why we selected patterns of length two and then applied the Huffman encoding technique for compression. This decreases the use of storage and increases the encoding and decoding speeds, because the detection of patterns of long lengths is relatively complex, and we normally obtain much lower frequencies of patterns. Figures 8.4 and 8.5, respectively, show the general block diagrams of the proposed encoding and decoding procedures. Additionally, the encoding and decoding procedures of the proposed method are given in Algorithms 8 and 9, respectively.

Algorithm 8: The proposed encoding procedure

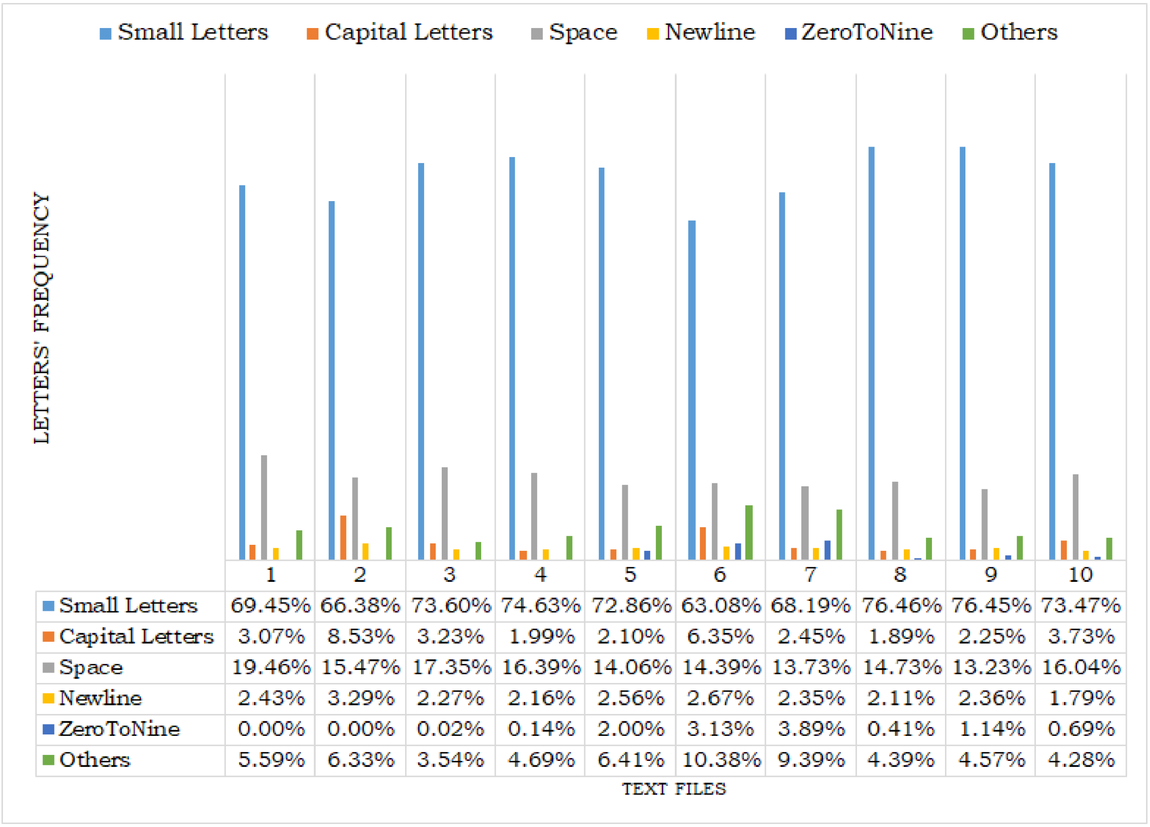
```
1  Read a text file as an input and calculate the length of the file (N);
2  Split the text into a set of small files of the same size (SS);
3  Apply the Burrows-Wheeler Transform on each text file separately and save the
     corresponding transform key. Set I =1.;
4  while $I \leq N$ do
5      if the number of the same consecutive characters is greater than three then
6          Store the tuple (key1, the character and their count) to ReducedText where
             count is converted into a character using the formula (counts+96);
7          I = I+count;
8      else if the number of the same consecutive characters is exactly three then
9          Store key2 and the character to ReducedText.;
10         I = I+3;
11     end
12     else
13         Save the character to the array ReducedText.;
14         I = I+1;
15     end
16 end
17 Find the specific patterns from ReducedText that have higher frequencies;
18 Apply Huffman encoding technique to get an encoded bit-stream;
19
```

8.4 Experimental Results and Analysis

Some experimental results are shown and explained in this section. We made a comparison with some other methods in order to show the usefulness of our proposed method. However, it is essential to determine the comparison parameters before making any comparisons. Here, the state-of-the-art techniques and the proposed method are compared based on the compression ratio (CR) that is calculated using Equation (8.1). It is a very important measurement criterion in this context [242]. Additionally, the encoding and decoding times are considered in the comparison.

$$CR = \frac{Original\ text\ size}{Compressed\ text\ size} \tag{8.1}$$

There are many lossless text compression algorithms, but we select PAQ8n, Deflate, Bzip2, Gzip, LZMA, LZW, and Brotli for comparison in this article, because those are the state-of-the-art techniques in this area, and Brotli is one of the best methods among them. We use some sample text files of different sizes from the UCI dataset for testing the aforementioned algorithms. Compression ratios are used in order to evaluate each method based on the sample texts. We apply state-of-the-art techniques and the proposed method on twenty different texts. Ta-



	1	2	3	4	5	6	7	8	9	10
Small Letters	69.45%	66.38%	73.60%	74.63%	72.86%	63.08%	68.19%	76.46%	76.45%	73.47%
Capital Letters	3.07%	8.53%	3.23%	1.99%	2.10%	6.35%	2.45%	1.89%	2.25%	3.73%
Space	19.46%	15.47%	17.35%	16.39%	14.06%	14.39%	13.73%	14.73%	13.23%	16.04%
Newline	2.43%	3.29%	2.27%	2.16%	2.56%	2.67%	2.35%	2.11%	2.36%	1.79%
ZeroToNine	0.00%	0.00%	0.02%	0.14%	2.00%	3.13%	3.89%	0.41%	1.14%	0.69%
Others	5.59%	6.33%	3.54%	4.69%	6.41%	10.38%	9.39%	4.39%	4.57%	4.28%

Figure 8.1: Comparison of letters' frequency in the texts.

Algorithm 9: The proposed decoding procedure

```
1  Apply Huffman decoding technique on the receive encoded bit-stream and store to
     PreReconstructedText1;
2  Set I=1 and calculate the length of the PreReconstructedText1 (Len);
3  while $I \leq Len$ do
4      if PreReconstructedText1[I]==Key1 then
5          Add (Int(PreReconstructedText1[I+2])-96) number of PreReconstructedText1
             [I+1] characters to PreReconstructedText2;
6          I = I+3;
7      else if PreReconstructedText1[I]==Key2 then
8          Add three PreReconstructedText1[I+1] characters to PreReconstructedText2;
9          I = I+2;
10     end
11     else
12         Add PreReconstructedText1[I] character to PreReconstructedText2;
13         I = I+1;
14     end
15 end
16 Split PreReconstructedText2 into a set of small files, where each file contains SS
     number of characters, and apply the inverse Burrows-Wheeler transform to each file
     with its corresponding TransformKey to get back the reconstructed text;
17
```

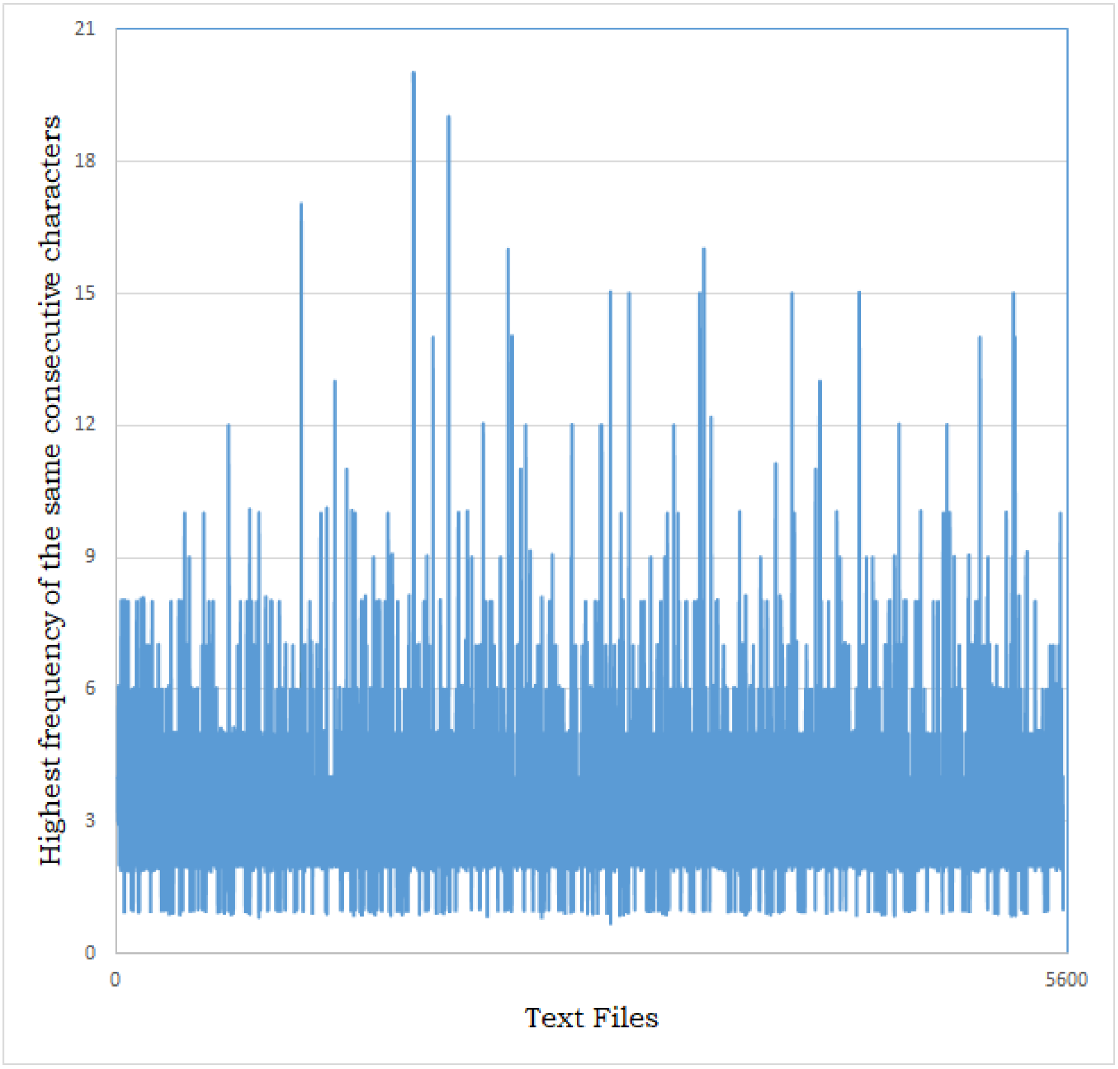


Figure 8.2: The highest frequencies of the same consecutive characters in the texts after the Burrows–Wheeler transform.

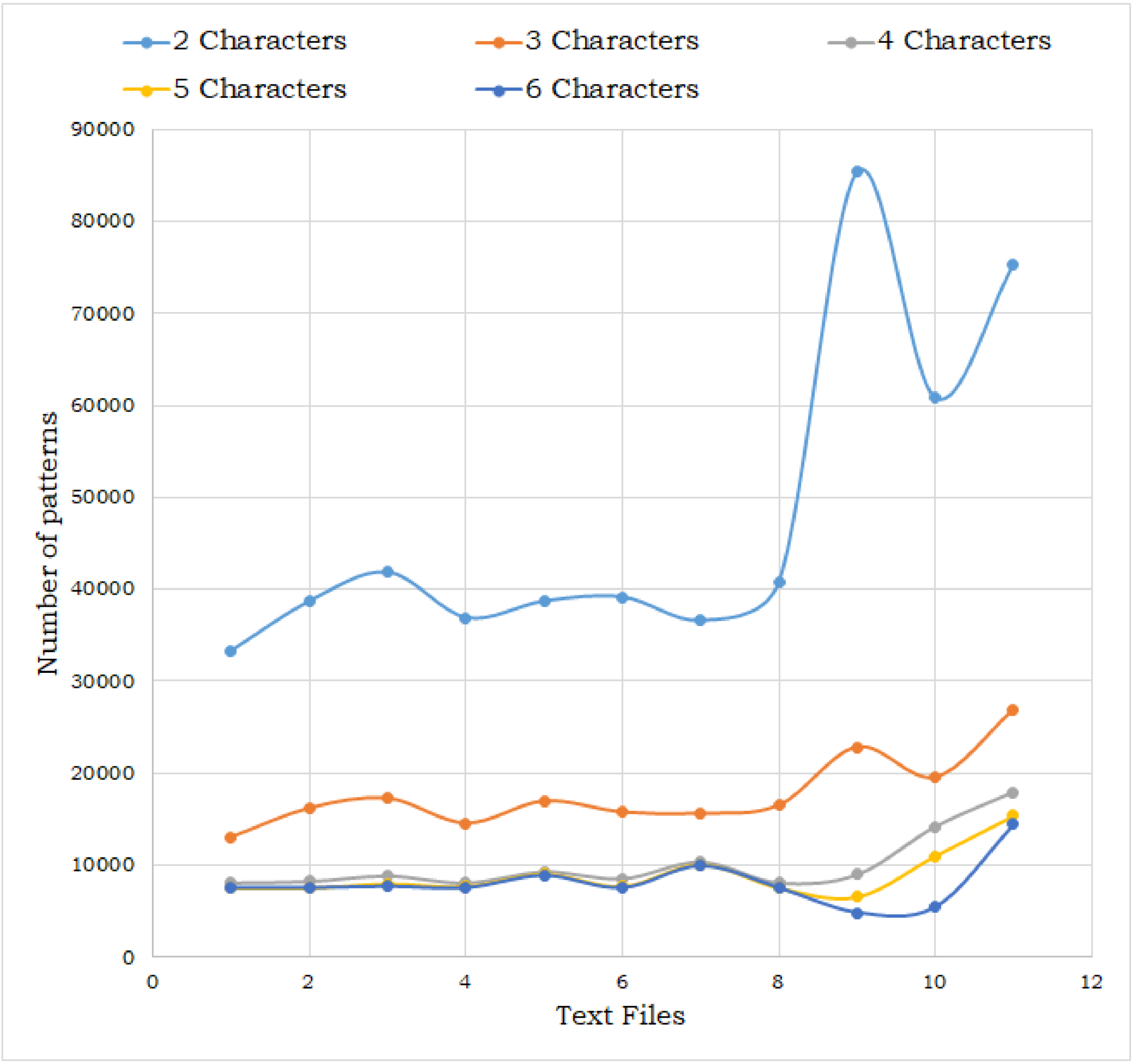


Figure 8.3: Frequency comparison of different patterns of different lengths.

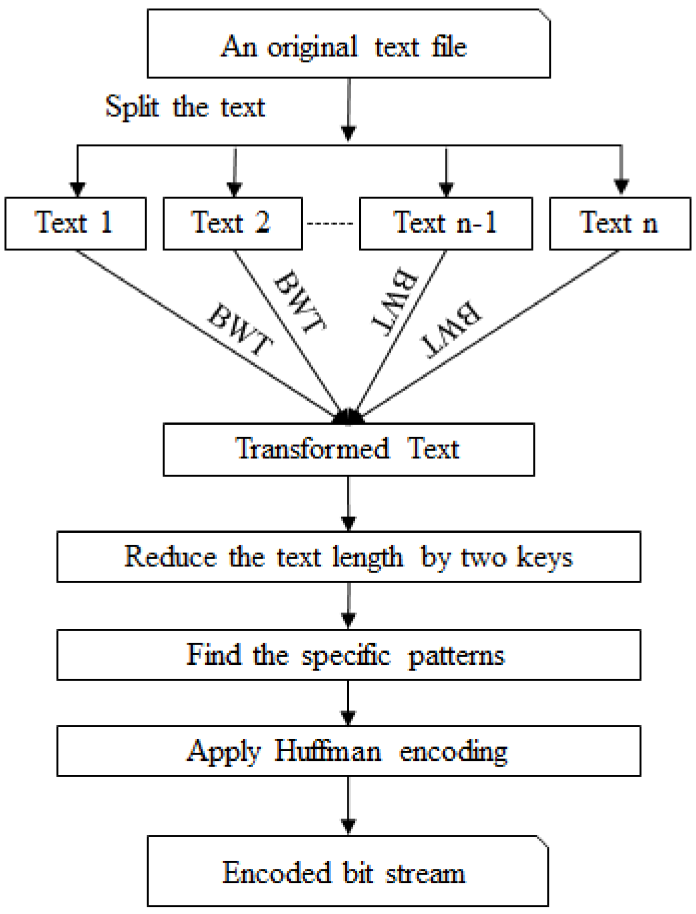


Figure 8.4: The general block diagram of the proposed encoding technique.

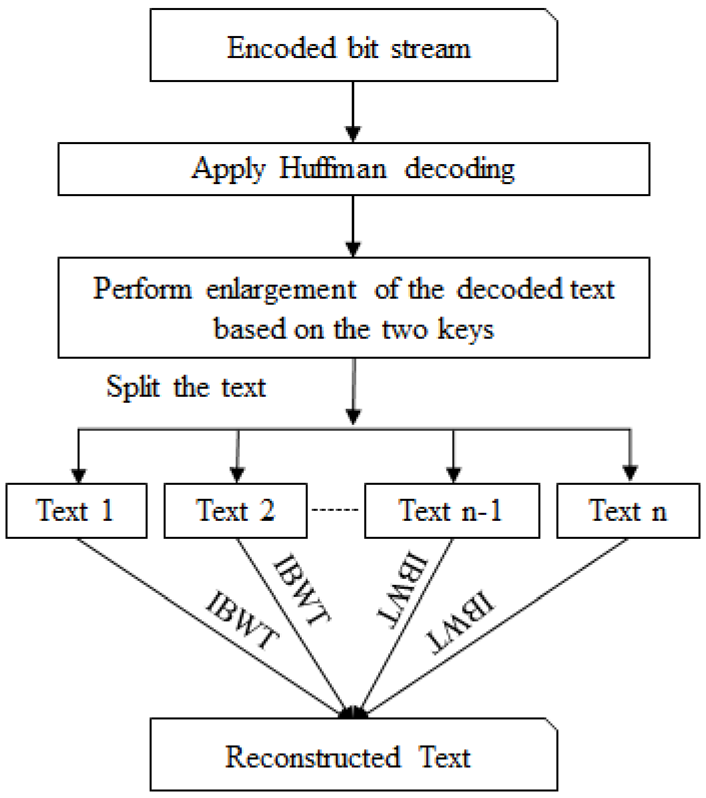


Figure 8.5: The general block diagram of the proposed decoding technique.

ble 8.1 shows the experimental results in terms of compression ratios of the texts and Figure 8.6 shows their graphical representation for quick comparison.

Table 8.1 shows that averagely LZW provides the lowest (1.288) compression ratio and Brotli the highest (1.667) among state-of-the-art techniques. Although PAQ8n provides 3.04%, 4.3%, 5.5%, and 3.91% better results than Brotli for the texts 3, 15, 16, and 20, respectively, Brotli shows 1.44%, 10.86%, 14.94%, 9.78%, 20.52%, and 22.74% more compression than PAQ8n, Deflate, Bzip2, Gzip, LZMA, and LZW, on average. It can be seen that the proposed technique provides better results, having a higher (1.884) compression ratio on average. Specifically, the proposed technique demonstrates, on average, a compression ratio 12.79% higher than PAQ8n, 21.13% higher than Deflate, 24.73% higher than Bzip2, 20.17% higher than Gzip, 31.63% higher than LZMA, 31.7% higher than LZW, and 11.52% higher than Brotli. We can see from Figure 6 that the compression ratio for the proposed technique is higher for every sample.

We also calculate the encoding and decoding times, which are shown in Figures 8.7 and 8.8, respectively. For encoding, on average, LZMA and Brotli take the highest (5.8915 s) and the lowest (0.0131 s) amounts of time, respectively, and the proposed technique takes 0.0375 s. PAQ8n and LZMA are 45.65% and 99.36% slower than the proposed coding technique. On the other hand, the proposed strategy takes 56.53%, 2.4%, 17.33%, 38.13%, and 65.07% more time than Deflate, Bzip2, Gzip, LZW, and Brotli, respectively. For decoding, on average, Brotli and LZMA take the lowest (0.007 s) and the highest (0.5896 s) amounts of time, respectively, and the proposed coding technique takes 0.0259 s. The proposed technique is 59.08% and 96.61% faster than PAQ8n, and LZMA, respectively; it is 49.81%, 47.1%, 7.34%, 27.8%, and 72.97% slower than Deflate, Bzip2, Gzip, LZW, and Brotli, respectively. In the case of both encoding and decoding time, we can conclude that our proposed coding method is faster than PAQ8n and LZMA and slower than the other methods that are mentioned in this article. Brotli performs the best out of the state-of-the-art methods in terms of compression ratios, encoding, and decoding times. However, our proposed method outperforms not only Brotli, but also the other state-of-the-art lossless text compression techniques that are mentioned in this article in terms of the compression ratio.

Text compression has two notable aspects based on its application: speed and storage efficiency. There are many applications, like Instant messaging, where speed is more important. On the other hand, a higher compression ratio is the primary concern for data storage applications. Because the proposed method provides more compression, it works better for the data storage applications. The compression ratio is inversely proportional to the total number of bits in a compressed file. Although the proposed method takes more time for encoding and decoding, a file compressed by the proposed method can be sent more quickly through a transmission media, because the number of bits in the file is less than in other files compressed by other methods. Steganography is a very well-known technique used for information hiding and is very important for Today's technology [218, 219]. Additionally, we can also use the proposed compression method with steganography when transferring a file securely over the Internet. To obtai a stego-text, we may first apply the steganography technique to a text file and then compress the stego-text by the proposed method to get a more secure text.

In this research work, we use the languages C++ and MATLAB (version 9.8.0.1323502 (R2020a)); CodeBlocks (20.03) and MATLAB are used as the coding environments. We also use an HP laptop with the Intel Core i3-3110M @2.40 GHz processor.

As a research direction, we can suggest from our investigation that Brotli is one of the best text compression methods. Brotli cannot provide satisfactory results for a large file compression due to its limited sliding window. However, it is a relatively fast compression method. If we can solve the sliding window problem satisfactorily while maintaining the same speed, Brotli will perform well from every point of view. On the other hand, our proposed method is somewhat slow. If we can increase its encoding and decoding speed, it will give better results.

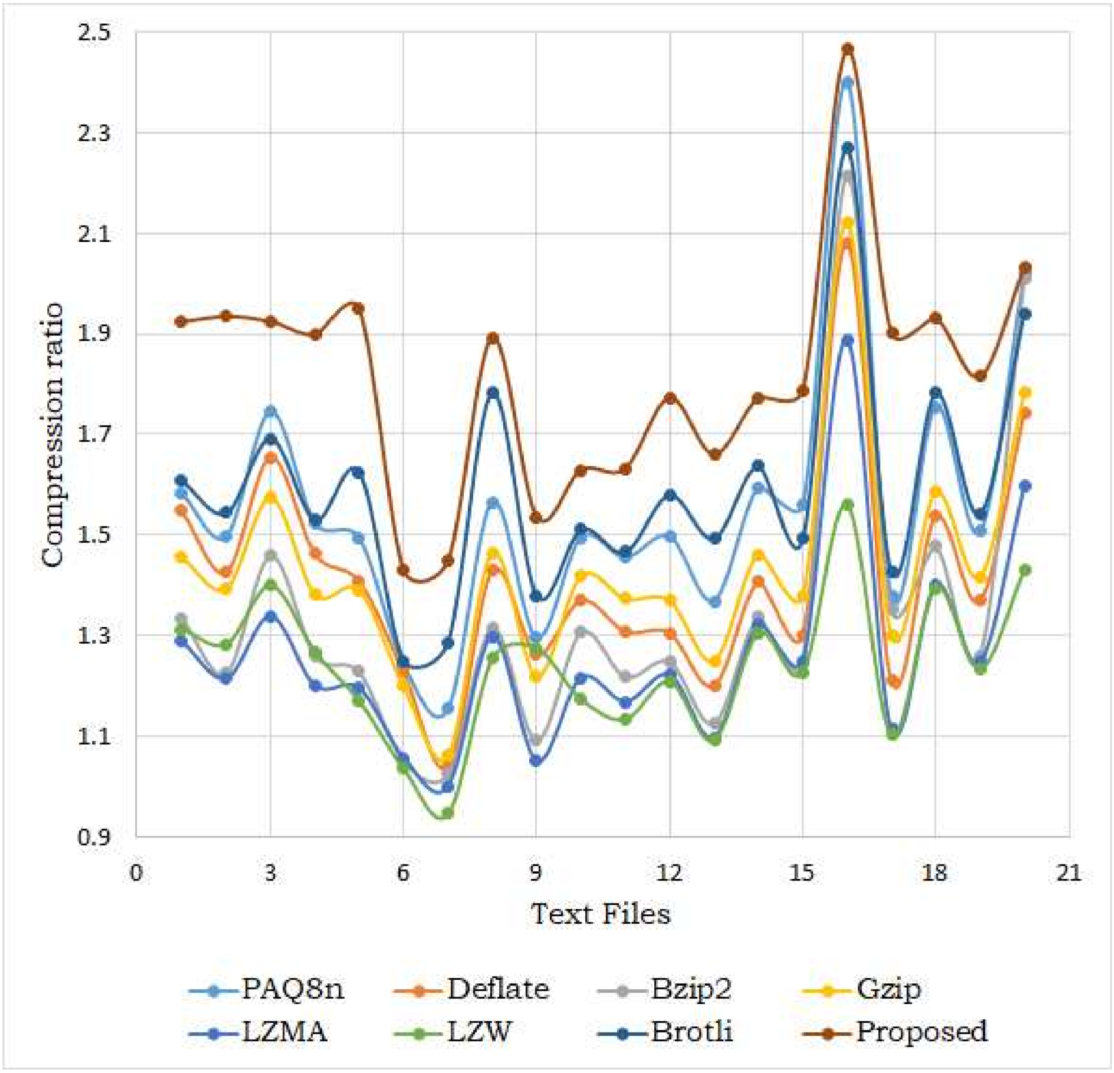


Figure 8.6: Graphical representation of the compression ratios.

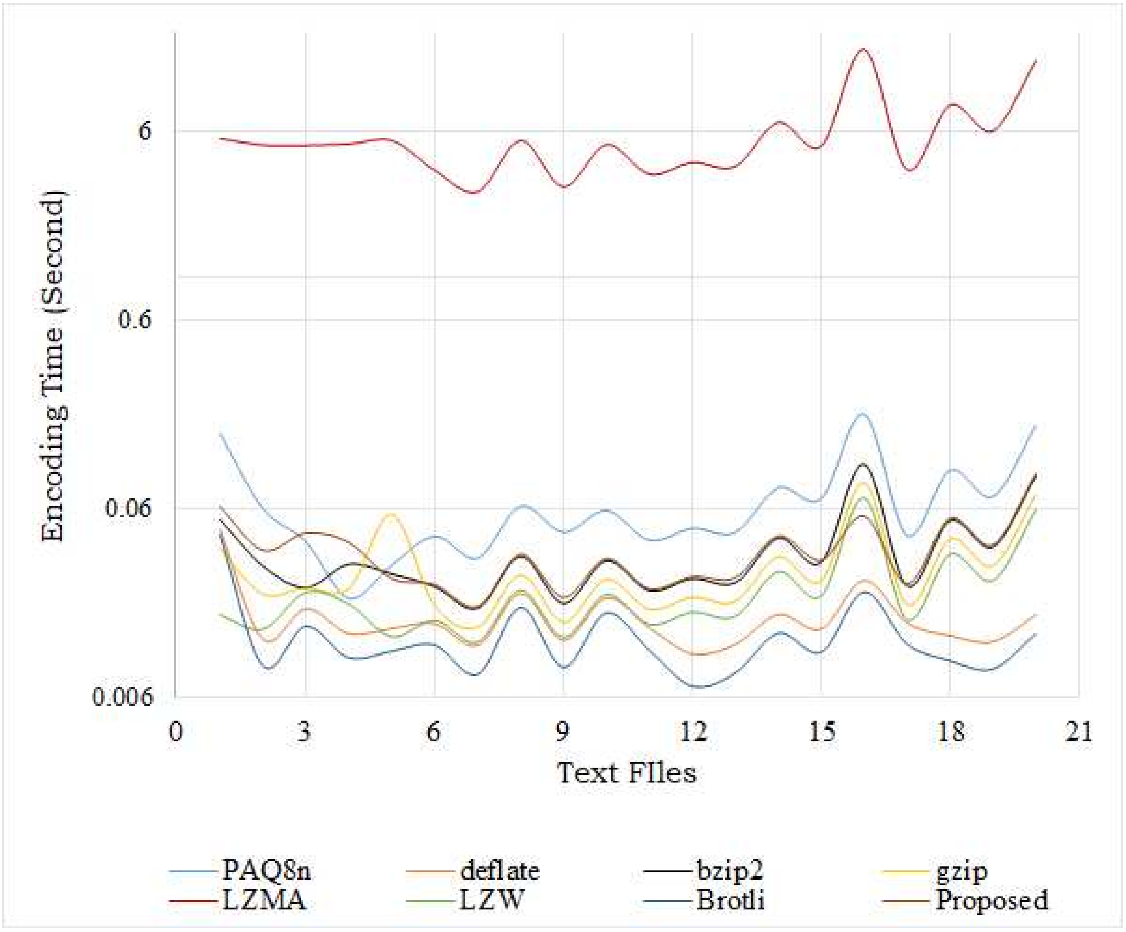


Figure 8.7: Encoding time comparison.

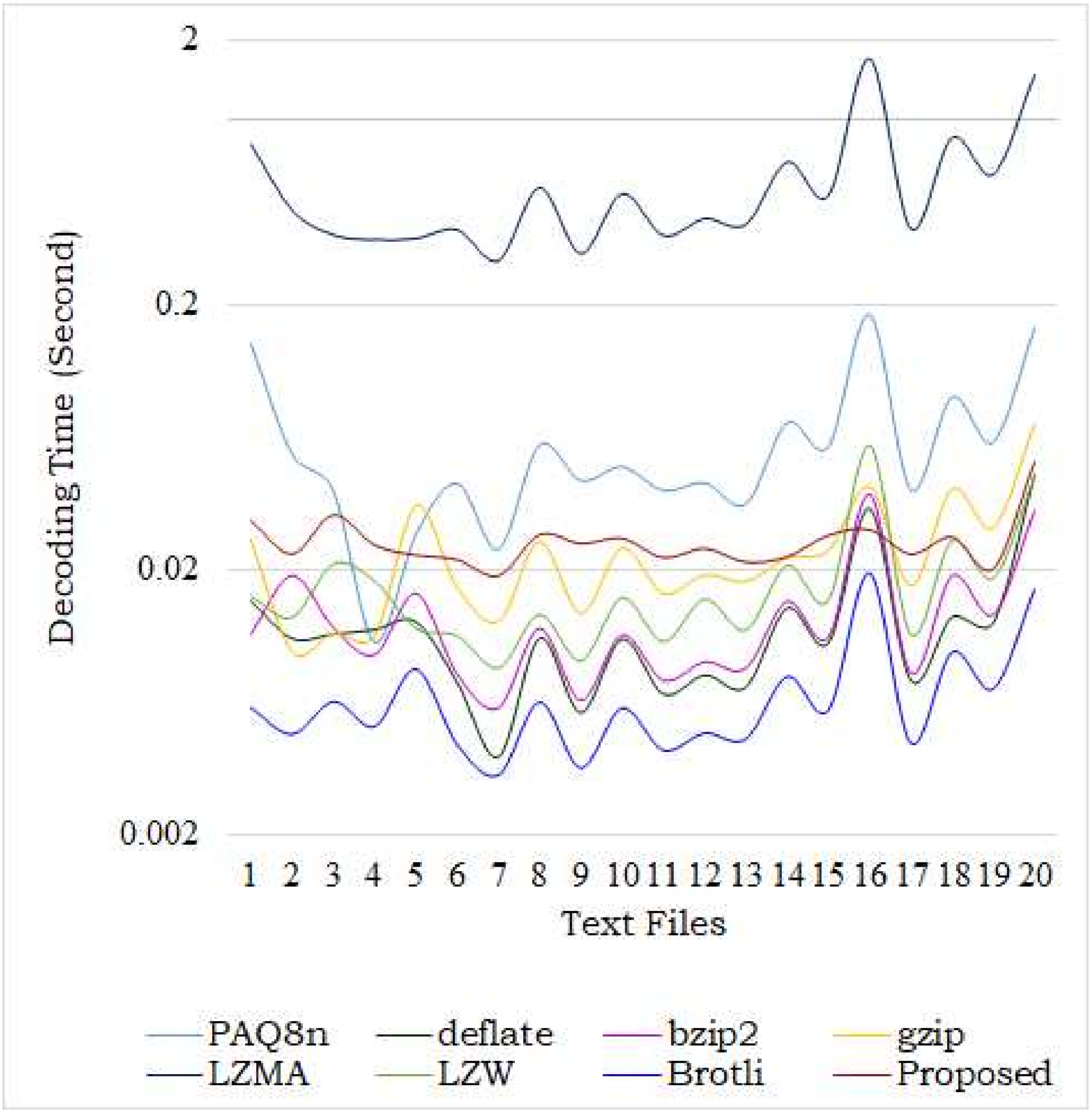


Figure 8.8: Decoding time comparison.

8.5 Summary

Lossless text compression is a more significant matter when there is a highly narrow-band communication channel and less storage available. We have proposed a completely lossless text compression while using the Burrows–Wheeler transform, two keys, and Huffman coding. What distinguishes our proposed method? First, to speed up the transformation, we split a large text file into sets of smaller files that ultimately increase the speed of compression. Second, we do not count all characters, which is done in run-length coding. We count only the characters that are repeated more than two times consecutively and replace the value of the letter count by a lowercase letter to increase the frequency of characters in the text, as each text contains the maximum number of lowercase letters. Third, we look for patterns of a certain length that have the highest frequency, so that we can get better results after applying Huffman coding.

The experimental outcomes show that the proposed method performs better than the seven algorithms that we compared it to: PAQ8n, Deflate, Bzip2, Gzip, LZMA, LZW, and Brotli. When used on the twenty sample texts, the proposed method gives an average of 21.66% higher compression than the methods that were described in this research.

One good aspect of our method is that we do not use any static dictionary, which helps to speed up the compression somewhat. Another special feature is that we find patterns of the same length. As a result, the complexity of finding patterns is minimal, and the highest frequency patterns are found, which leads to a better compression ratio.

Table 8.1: Comparison among compression ratios

Texts	**PAQ8n**	**Deflate**	**Bzip2**	**Gzip**	**LZMA**	**LZW**	**Brotli**	**Proposed**
1	1.582	1.548	1.335	1.455	1.288	1.313	1.608	**1.924**
2	1.497	1.427	1.226	1.394	1.214	1.283	1.544	**1.935**
3	1.745	1.655	1.46	1.574	1.338	1.399	1.692	**1.925**
4	1.523	1.463	1.261	1.382	1.2	1.268	1.531	**1.899**
5	1.493	1.408	1.228	1.39	1.195	1.17	1.625	**1.949**
6	1.242	1.228	1.051	1.199	1.057	1.036	1.25	**1.429**
7	1.154	1.04	1.026	1.061	1	0.946	1.287	**1.448**
8	1.566	1.43	1.316	1.465	1.298	1.254	1.783	**1.893**
9	1.295	1.265	1.092	1.219	1.05	1.275	1.38	**1.536**
10	1.495	1.371	1.307	1.419	1.216	1.174	1.511	**1.629**
11	1.455	1.309	1.219	1.373	1.168	1.134	1.466	**1.632**
12	1.497	1.306	1.249	1.37	1.222	1.209	1.58	**1.773**
13	1.369	1.201	1.126	1.25	1.097	1.092	1.493	**1.66**
14	1.595	1.407	1.336	1.462	1.321	1.305	1.637	**1.773**
15	1.559	1.302	1.243	1.38	1.249	1.227	1.492	**1.788**
16	2.401	2.082	2.214	2.121	1.888	1.559	2.269	**2.466**
17	1.38	1.211	1.353	1.302	1.113	1.103	1.428	**1.903**
18	1.755	1.537	1.477	1.585	1.401	1.394	1.782	**1.931**
19	1.507	1.37	1.261	1.417	1.247	1.234	1.542	**1.815**
20	2.02	1.744	2.01	1.783	1.596	1.43	1.941	**2.033**
Average	**1.643**	**1.486**	**1.418**	**1.504**	**1.325**	**1.288**	**1.667**	**1.884**

Chapter 9

Conclusions and Future Work

Lossless still image compression is a topic of intense research interest in the field of computing, especially for applications that rely on low bandwidth connections and limited storage. In this thesis, we contributed to the theory and practice of lossless data compression in some ways. In the following, We give a brief description of the problems considered in this thesis and our proposed solutions.

1. In Chapter 2, we focused on three things. First of all, we explained the basic concept of how to transform a continuous tone image to its digital form. Secondly, we explained in detail how to compress an image. Finally, we explained the measurement standards used to evaluate a data compression algorithm.
2. In Chapter 3, we studied the transformation techniques used for data compression and explained the techniques based on numeric data for clear understanding.
3. In Chapter 4, we studied the state-of-the-art lossless data compression techniques. In particular, the encoding and decoding procedures of the techniques are explained in detail.
4. In Chapter 5, we studied the entropy coding methods used in each of the advanced data compression techniques. Since, choosing an entropy coding strategy for data compression is a big challenge. We explained the methods based on a common numeric data set. We also gave an extensive analysis based on the experimental results of some images and recommended the best entropy coding technique. We finally showed the limitations of the algorithms and demonstrated which part of the algorithms needs to be improved.
5. In Chapter 6, we proposed a mathematical model to select an optimal lossless image compression technique is proposed in this chapter. This chapter shows that each algorithm was evaluated based on a specific parameter in each research work. However, the performance of a lossless image compression algorithm depends on all parameters (bpp, encoding and decoding time) and does not singly depend on any of them. Therefore, the proposed method predicts a better lossless image compression algorithm for any combination of parameters and provides the actual impact of each algorithm.
6. The PCBMS model gives a better prediction to select a better lossless image compression method. However, a better prediction depends on making a good balance between compression ratio, encoding, and decoding times. Therefore, we proposed an alternative approach to PCBMS in Chapter 7. The proposed method can balance the parameters better than PCBMS and give more accurate predictions.
7. In Chapter 8, we studied the text compression method. When run-length coding is used in terms of data compression, it increases the number of unique symbols during the coding of characters' length. As a result, the compression ratio is reduced. To solve the problem,

we proposed a key-based coding procedure in place of run-length coding that increases the compression ratio.

Some algorithm provides better compression ratios. Others may be able to speed up encoding or decoding or both. One algorithm gives better results for one type of image, another algorithm gives better results for another type of image. Another problem is that an algorithm can give better performance for an image in a similar type of image. In contrast, another algorithm may give better results for another image of the same type. So, finding the best algorithm for each individual image is a big challenge. For this issue, designing a deep convolutional neural network-based model will be an interesting direction for future work.

References

[1] M. J. Weinberger, G. Seroussi, and G. Sapiro, "Loco-i: A low complexity, context-based, lossless image compression algorithm," in *Proceedings of Data Compression Conference-DCC'96*. IEEE, 1996, pp. 140–149.

[2] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal processing magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[3] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Practical full resolution learned lossless image compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 629–10 638.

[4] S. Cao, C.-Y. Wu, and P. Krähenbühl, "Lossless image compression through super-resolution," *arXiv preprint arXiv:2004.02872*, 2020.

[5] A. Alarabeyyat, S. Al-Hashemi, T. Khdour, M. H. Btoush, S. Bani-Ahmad, R. Al-Hashemi, S. Bani-Ahmad *et al.*, "Lossless image compression technique using combination methods," *Journal of Software Engineering and Applications*, vol. 5, no. 10, p. 752, 2012.

[6] M. U. A. Ayoobkhan, E. Chikkannan, K. Ramakrishnan, and S. B. Balasubramanian, "Prediction-based lossless image compression," in *International Conference on ISMAC in Computational Vision and Bio-Engineering*. Springer, 2018, pp. 1749–1761.

[7] X. O. Zhao and Z. H. He, "Lossless image compression using super-spatial structure prediction," *IEEE Signal Processing Letters*, vol. 17, no. 4, pp. 383–386, 2010.

[8] B. Xiao, G. Lu, Y. Zhang, W. Li, and G. Wang, "Lossless image compression based on integer discrete tchebichef transform," *Neurocomputing*, vol. 214, pp. 587–593, 2016.

[9] Z. Zuo, X. Lan, L. Deng, S. Yao, and X. Wang, "An improved medical image compression technique with lossless region of interest," *Optik*, vol. 126, no. 21, pp. 2825–2831, 2015.

[10] J. Li, "An improved wavelet image lossless compression algorithm," *Optik*, vol. 124, no. 11, pp. 1041–1044, 2013.

[11] R. N. Kumar, B. Jagadale, and J. Bhat, "A lossless image compression algorithm using wavelets and fractional fourier transform," *SN Applied Sciences*, vol. 1, no. 3, pp. 1–8, 2019.

[12] T. Strutz, "Context-based predictor blending for lossless color image compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 4, pp. 687–695, 2015.

[13] B. Rusyn, O. Lutsyk, Y. Lysak, A. Lukenyuk, and L. Pohreliuk, "Lossless image compression in the remote sensing applications," in *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2016, pp. 195–198.

[14] G. Al-Khafaji and L. E. George, "Fast lossless compression of medical images based on polynomial," *International Journal of Computer Applications*, vol. 70, no. 15, 2013.

[15] C. Perra, "Lossless plenoptic image compression using adaptive block differential prediction," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1231–1234.

[16] R. R. S. Tomar and K. Jain, "Lossless image compression using differential pulse code modulation and its application," in *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2015, pp. 397–400.

[17] A. D. Wilson, "Fast lossless depth image compression," in *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*, 2017, pp. 100–105.

[18] H. Zhang, X.-q. Wang, Y.-j. Sun, and X.-y. Wang, "A novel method for lossless image compression and encryption based on lwt, spiht and cellular automata," *Signal Processing: Image Communication*, vol. 84, p. 115829, 2020.

[19] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, 2018.

[20] T. Bruylants, A. Munteanu, and P. Schelkens, "Wavelet based volumetric medical image compression," *Signal processing: Image communication*, vol. 31, pp. 112–133, 2015.

[21] D. Venugopal, S. Mohan, and S. Raja, "An efficient block based lossless compression of medical images," *Optik*, vol. 127, no. 2, pp. 754–758, 2016.

[22] L. Yang, X. He, G. Zhang, L. Qing, and T. Che, "A low complexity block-based adaptive lossless image compression," *Optik*, vol. 124, no. 24, pp. 6545–6552, 2013.

[23] R. Sumalatha and M. Subramanyam, "Hierarchical lossless image compression for telemedicine applications," *Procedia Computer Science*, vol. 54, pp. 838–848, 2015.

[24] A. Masmoudi, W. Puech, and A. Masmoudi, "An improved lossless image compression based arithmetic coding using mixture of non-parametric distributions," *Multimedia Tools and Applications*, vol. 74, no. 23, pp. 10 605–10 619, 2015.

[25] A. Abdollahi, N. Bruce, S. Kamali, and R. Karim, "Lossless image compression using list update algorithms," in *International Symposium on String Processing and Information Retrieval*. Springer, 2019, pp. 16–34.

[26] A. Khan, A. Khan, M. Khan, and M. Uzair, "Lossless image compression: application of bi-level burrows wheeler compression algorithm (bbwca) to 2-d data," *Multimedia Tools and Applications*, vol. 76, no. 10, pp. 12 391–12 416, 2017.

[27] M. Rahman, M. Hamada *et al.*, "Lossless image compression techniques: A state-of-the-art survey," *Symmetry*, vol. 11, no. 10, p. 1274, 2019.

[28] M. A. Rahman and M. Hamada, "PCBMS: A model to selectan optimal lossless image compression technique," *IEEE Access, DOI:10.1109/AC-CESS.2021.3137345*, 2021.

[29] M. Rahman, M. Hamada, J. Shin *et al.*, "The impact of state-of-the-art techniques for lossless still image compression," *Electronics*, vol. 10, no. 3, p. 360, 2021.

[30] M. Rahman, M. Hamada *et al.*, "Burrows–wheeler transform based lossless text compression using keys and huffman coding," *Symmetry*, vol. 12, no. 10, p. 1654, 2020.

[31] M. A. Rahman and M. Hamada, “Lossless text compression using gpt-2 language model and huffman coding,” in *In Proceedings of The 2021 3rd ETLTC - ACM Chapter International Conference on Information and Communications Technology*. ACM, 2021.

[32] R. C. Gonzalez, S. L. Eddins, and R. E. Woods, *Digital image publishing using MATLAB*. Prentice Hall, 2004.

[33] E. R. Dougherty, *Digital image processing methods*. CRC Press, 2020.

[34] M. Kitamura, D. Shirai, K. Kaneko, T. Murooka, T. Sawabe, T. Fujii, and A. Takahara, “Beyond 4k: 8k 60p live video streaming to multiple sites,” *Future Generation Computer Systems*, vol. 27, no. 7, pp. 952–959, 2011.

[35] T. Yamashita and K. Mitani, “8k extremely-high-resolution camera systems,” *Proceedings of the IEEE*, vol. 101, no. 1, pp. 74–88, 2012.

[36] U. TODAY, “Usatoday.com,” https://www.usatoday.com/story/tech/columnist/komando/2012/11/30/komando-computer-storage/1726835/, 2020, [Accessed 14 September 2020].

[37] Statista, “Seagate Average HDD Capacity Worldwide 2015-2020,” https://www.statista.com/statistics/795748/worldwide-seagate-average-hard-disk-drive-capacity/, 2020, [Accessed 14 September 2020].

[38] D. Cunningham, B. Lane, and W. Lane, *Gigabit ethernet networking*. Macmillan Publishing Co., Inc., 1999.

[39] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, “On the treeness of internet latency and bandwidth,” in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, 2009, pp. 61–72.

[40] M. Rabbani and P. W. Jones, *Digital image compression techniques*. SPIE press, 1991, vol. 7.

[41] M. Nelson and J.-L. Gailly, “The data compression book 2nd edition,” *M & T Books, New York, NY*, 1995.

[42] G. Padmaja and P. Nirupama, “Analysis of various image compression techniques,” *ARPN Journal of Science and Technology*, vol. 2, no. 4, pp. 371–376, 2012.

[43] M. Barni, *Document and Image compression*. CRC press, 2018.

[44] S. Dhawan, “A review of image compression and comparison of its algorithms,” *International Journal of electronics & Communication technology*, vol. 2, no. 1, pp. 22–26, 2011.

[45] J. A. Storer, *Image and text compression*. Springer Science & Business Media, 2012, vol. 176.

[46] S. E. Umbaugh, *Computer imaging: digital image analysis and processing*. CRC press, 2005.

[47] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.

[48] X. Lu, H. Wang, W. Dong, F. Wu, Z. Zheng, and G. Shi, “Learning a deep vector quantization network for image compression,” *IEEE Access*, vol. 7, pp. 118 815–118 825, 2019.

[49] Y. Zhang, H. Cao, H. Jiang, and B. Li, “Visual distortion sensitivity modeling for spatially adaptive quantization in remote sensing image compression,” *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 4, pp. 723–727, 2013.

[50] C. Cai, L. Chen, X. Zhang, and Z. Gao, “End-to-end optimized roi image compression,” *IEEE Transactions on Image Processing*, vol. 29, pp. 3442–3457, 2019.

[51] S. Liu, W. Bai, N. Zeng, and S. Wang, “A fast fractal based compression for mri images,” *IEEE Access*, vol. 7, pp. 62 412–62 420, 2019.

[52] Z. Wang and A. C. Bovik, “A universal image quality index,” *IEEE signal processing letters*, vol. 9, no. 3, pp. 81–84, 2002.

[53] D.-s. Huang, “Radial basis probabilistic neural networks: Model and application,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 13, no. 07, pp. 1083–1101, 1999.

[54] T. Ishigaki, S. Sakuma, M. Ikeda, Y. Itoh, M. Suzuki, and S. Iwai, “Clinical evaluation of irreversible image compression: analysis of chest imaging with computed radiography.” *Radiology*, vol. 175, no. 3, pp. 739–743, 1990.

[55] S. Lloyd, “Least squares quantization in pcm,” *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[56] Z. Zhang, G. Dai, X. Liang, S. Yu, L. Li, and Y. Xie, “Can signal-to-noise ratio perform as a baseline indicator for medical image quality assessment,” *IEEE Access*, vol. 6, pp. 11 534–11 543, 2018.

[57] D. M. Chandler and S. S. Hemami, “Vsnr: A wavelet-based visual signal-to-noise ratio for natural images,” *IEEE transactions on image processing*, vol. 16, no. 9, pp. 2284–2298, 2007.

[58] V. Kuperman, *Magnetic resonance imaging: physical principles and applications*. Elsevier, 2000.

[59] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, “Comparison of the coding efficiency of video coding standards—including high efficiency video coding (hevc),” *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1669–1684, 2012.

[60] R. Seymour, D. Stewart, and J. Ming, “Comparison of image transform-based features for visual speech recognition in clean and corrupted videos,” *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–9, 2007.

[61] V. K. Bairagi, A. M. Sapkal, and M. Gaikwad, “The role of transforms in image compression,” *Journal of The Institution of Engineers (India): Series B*, vol. 94, no. 2, pp. 135–140, 2013.

[62] N. Astaf’eva, “Wavelet analysis: basic theory and some applications,” *Physics-Uspekhi*, vol. 39, no. 11, p. 1085, 1996.

[63] D. Popov, A. Gapochkin, and A. Nekrasov, “An algorithm of daubechies wavelet transform in the final field when processing speech signals,” *Electronics*, vol. 7, no. 7, p. 120, 2018.

[64] M. Burrows and D. Wheeler, “A block-sorting lossless data compression algorithm,” in *Digital SRC Research Report*. Citeseer, 1994.

[65] H. M. Rahman, Md Atiqur and R. M. Asfaqur, “Text compression based on an alternative approachof run-length coding using burrows-wheelertransform and arithmetic coding,” in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC).* IEEE, 2021.

[66] En.wikipedia.org, “Burrows–Wheeler Transform,” https://en.wikipedia.org/wiki/Burrows_Wheeler_transform/, 2020, [Accessed 27 May 2020].

[67] S.-D. Kim, J.-H. Lee, and J.-K. Kim, “A new chain-coding algorithm for binary images using run-length codes,” *Computer vision, Graphics, and Image processing*, vol. 41, no. 1, pp. 114–128, 1988.

[68] B. Žalik, D. Mongus, and N. Lukač, “A universal chain code compression method,” *Journal of Visual Communication and Image Representation*, vol. 29, pp. 8–15, 2015.

[69] M. A. Rahman and M. Hamada, “A semi-lossless image compression procedure using a lossless mode of jpeg,” in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC).* IEEE, 2019, pp. 143–148.

[70] C. E. Shannon, “A mathematical theory of communication,” *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[71] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Resonance*, vol. 11, no. 2, pp. 91–99, 2006.

[72] T. Xue, Y. Zhang, Y. Shen, Z. Zhang, X. You, and C. Zhang, “Adaptive spatial modulation combining bch coding and huffman coding,” in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP).* IEEE, 2018, pp. 1–5.

[73] Z. Yan-li, F. Xiao-ping, L. Shao-qiang, and X. Zhe-yuan, “Improved lzw algorithm of lossless data compression for wsn,” in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 4. IEEE, 2010, pp. 523–527.

[74] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[75] G. G. Langdon, “An introduction to arithmetic coding,” *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[76] L. Sasilal and V. Govindan, “Arithmetic coding-a reliable implementation,” *International Journal of Computer Applications*, vol. 73, no. 7, 2013.

[77] J.-J. Ding and I.-H. Wang, “Improved frequency table adjusting algorithms for context-based adaptive lossless image coding,” in *2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW).* IEEE, 2016, pp. 1–2.

[78] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[79] D. Salomon, *Data compression: the complete reference.* Springer Science & Business Media, 2004.

[80] D. A. Clunie, “Lossless compression of grayscale medical images: effectiveness of traditional and state-of-the-art approaches,” in *Medical Imaging 2000: PACS Design and Evaluation: Engineering and Clinical Issues*, vol. 3980. International Society for Optics and Photonics, 2000, pp. 74–84.

[81] J. Kim and C.-M. Kyung, “A lossless embedded compression using significant bit truncation for hd video coding,” *IEEE Transactions on Circuits and Systems for video technology*, vol. 20, no. 6, pp. 848–860, 2010.

[82] M. Sharma *et al.*, “Compression using huffman coding,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 5, pp. 133–141, 2010.

[83] M. Kato, “Motion video coding with adaptive precision for dc component coefficient quantization and variable length coding,” Sep. 24 1996, uS Patent 5,559,557.

[84] C. Lamorahan, B. Pinontoan, and N. Nainggolan, “Data compression using shannon-fano algorithm,” *d’CARTESIAN*, vol. 2, no. 2, pp. 10–17, 2013.

[85] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard.* Springer Science & Business Media, 1992.

[86] N. S. Jayant and P. Noll, “Digital coding of waveforms: principles and applications to speech and video,” *Englewood Cliffs, NJ*, pp. 115–251, 1984.

[87] M. J. Weinberger, G. Seroussi, and G. Sapiro, “The loco-i lossless image compression algorithm: Principles and standardization into jpeg-ls,” *IEEE Transactions on Image processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[88] I. Ueno and F. Ono, “Proposed modification of loco-i for its improvement of the performance." iso,” IEC JTC1/SC29/WG1 document, Tech. Rep.

[89] M. Weinberger, G. Seroussi, and G. Sapiro, “Fine-tuning the baseline." iso,” IEC JTC1/SC29/WG1 document, Tech. Rep.

[90] M. J. Weinberger, G. Seroussi, and G. Sapiro, “Palettes and sample mapping in jpeg-ls." iso,” IEC JTC1/SC29/WG1 document, Tech. Rep.

[91] M. Weinberger, G. Seroussi, G. Sapiro, and E. Ordentlich, “Jpeg-ls with limited-length code words." iso,” IEC JTC1/SC29/WG1 document, Tech. Rep.

[92] J. J. Rissanen, “Generalized kraft inequality and arithmetic coding,” *IBM Journal of research and development*, vol. 20, no. 3, pp. 198–203, 1976.

[93] J. Rissanen and G. Langdon, “Universal modeling and coding,” *IEEE Transactions on Information Theory*, vol. 27, no. 1, pp. 12–23, 1981.

[94] N. Merhav, G. Seroussi, and M. J. Weinberger, *Lossless compression for sources with two-sided geometric distributions.* Citeseer, 1998.

[95] M. J. Weinberger, G. Seroussi, and G. Sapiro, “From logo-i to the jpeg-ls standard,” in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 4. IEEE, 1999, pp. 68–72.

[96] N. D. Memon, X. Wu, V. Sippy, and G. Miller, “Interband coding extension of the new lossless jpeg standard,” in *Visual Communications and Image Processing’97*, vol. 3024. International Society for Optics and Photonics, 1997, pp. 47–58.

[97] G. Roelofs, *PNG: the definitive guide.* O’Reilly Media, 1999.

[98] C. Wilbur, “Png: The definitive guide,” *Journal of Computing in Higher Education*, vol. 12, no. 2, pp. 94–97, 2001.

[99] Libpng.org, "PNG specification: Filter Algorithms," http://www.libpng.org/pub/png/spec/1.2/PNG-Filters.html, 2020, [Accessed 5 October 2020].

[100] A. W. Paeth, "Image file compression made easy," in *Graphics Gems II.* Elsevier, 1991, pp. 93–100.

[101] X. Wu, "An algorithmic study on lossless image compression," in *Proceedings of Data Compression Conference-DCC'96.* IEEE, 1996, pp. 150–159.

[102] X. Wu and N. Memon, "Calic-a context based adaptive lossless image codec," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 4. IEEE, 1996, pp. 1890–1893.

[103] Web.archive.org, "Diary Of An x264 Developer » H.264 and VP8 for still image coding: WebP," https://web.archive.org/web/20150319214453/http://x264dev.multimedia.cx/archives/541, 2021, [Accessed 30 March 2021].

[104] M. Pintus, G. Ginesu, L. Atzori, and D. D. Giusto, "Objective evaluation of webp image compression efficiency," in *International Conference on Mobile Multimedia Communications.* Springer, 2011, pp. 252–265.

[105] Z. Si and K. Shen, "Research on the webp image format," in *Advanced Graphic Communications, Packaging Technology and Materials.* Springer, 2016, pp. 271–277.

[106] H. Singh, *Practical Machine Learning and Image Processing.* Springer, 2019.

[107] G. Ginesu, M. Pintus, and D. D. Giusto, "Objective assessment of the webp image coding algorithm," *Signal Processing: Image Communication*, vol. 27, no. 8, pp. 867–874, 2012.

[108] G. Developers, "Compression Techniques | Webp | Google Developers," https://developers.google.com/speed/webp/docs/compression, 2021, [Accessed 19 January 2021].

[109] J. Sneyers and P. Wuille, "Flif: Free lossless image format based on maniac compression," in *2016 IEEE International Conference on Image Processing (ICIP).* IEEE, 2016, pp. 66–70.

[110] N. Soferman, "Flif, the new lossless image format that outperforms png, webp and bpg," 2021.

[111] Flif.info, "FLIF - Free Lossless Image Format," https://flif.info/, 2021, [Accessed 2 January 2021].

[112] ——, "FLIF - Example," https://flif.info/example.html, 2021, [Accessed 2 January 2021].

[113] ——, "FLIF - Software," https://flif.info/software.html, 2021, [Accessed 2 January 2021].

[114] M. Boliek, "Jpeg2000 part i final draft international standard," *(ISO/IEC FDIS15444-1), ISO/IEC JTC1/SC29/WG1 N1855*, 2000.

[115] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The jpeg2000 still image coding system: an overview," *IEEE transactions on consumer electronics*, vol. 46, no. 4, pp. 1103–1127, 2000.

[116] P. Schelkens, A. Skodras, and T. Ebrahimi, *The JPEG 2000 suite.* John Wiley & Sons, 2009, vol. 15.

[117] D. Santa-Cruz, T. Ebrahimi, J. Askelof, M. Larsson, and C. A. Christopoulos, "Jpeg 2000 still image coding versus other standards," in *Applications of digital image processing XXIII*, vol. 4115. International Society for Optics and Photonics, 2000, pp. 446–454.

[118] H. R. Sheikh, A. C. Bovik, and L. Cormack, "No-reference quality assessment using natural scene statistics: Jpeg2000," *IEEE Transactions on image processing*, vol. 14, no. 11, pp. 1918–1927, 2005.

[119] Z. P. Sazzad, Y. Kawayoke, and Y. Horita, "No reference image quality assessment for jpeg2000 based on spatial features," *Signal Processing: Image Communication*, vol. 23, no. 4, pp. 257–268, 2008.

[120] C. S. Swartz, *Understanding digital cinema: a professional handbook*. Taylor & Francis, 2005.

[121] M. Rabbani, "Jpeg2000: Image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, vol. 11, no. 2, p. 286, 2002.

[122] T. Kim, H. M. Kim, P.-S. Tsai, and T. Acharya, "Memory efficient progressive rate-distortion algorithm for jpeg 2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 1, pp. 181–187, 2005.

[123] Z. Liu, L. J. Karam, and A. B. Watson, "Jpeg2000 encoding with perceptual distortion control," *IEEE transactions on image processing*, vol. 15, no. 7, pp. 1763–1778, 2006.

[124] J. Zhang and T. M. Le, "A new no-reference quality metric for jpeg2000 images," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 743–750, 2010.

[125] A. C. Bovik, *The essential guide to video processing*. Academic Press, 2009.

[126] M. Unser and T. Blu, "Mathematical properties of the jpeg2000 wavelet filters," *IEEE transactions on image processing*, vol. 12, no. 9, pp. 1080–1090, 2003.

[127] B. Crow, "Bill Crow's Digital imaging & photography blog," https://docs.microsoft.com/en-us/archive/blogs/billcrow/, 2020, [Accessed 8 October 2020].

[128] F. Dufaux, G. J. Sullivan, and T. Ebrahimi, "The jpeg xr image coding standard [standards in a nutshell]," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 195–204, 2009.

[129] F. De Simone, L. Goldmann, V. Baroncini, and T. Ebrahimi, "Subjective evaluation of jpeg xr image compression," in *Applications of Digital Image Processing XXXII*, vol. 7443. International Society for Optics and Photonics, 2009, p. 74430L.

[130] C. Tu, S. Srinivasan, G. J. Sullivan, S. Regunathan, and H. S. Malvar, "Low-complexity hierarchical lapped transform for lossy-to-lossless image coding in jpeg xr/hd photo," in *Applications of Digital Image Processing XXXI*, vol. 7073. International Society for Optics and Photonics, 2008, p. 70730C.

[131] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre-and post-filtering," *IEEE Transactions on Signal Processing*, vol. 51, no. 6, pp. 1557–1571, 2003.

[132] S. Zimmerman, "A Look At AV1 And The Future Of Video Codecs: Google's Answer To HEVC," https://www.xda-developers.com/av1-future-video-codecs-google-hevc/, 2021, [Accessed 2 January 2021].

[133] J. Ozer, "What Is VP9?. [online] Streaming Media Magazine," https://www.streamingmedia.com/Articles/Editorial/-111334.aspx, 2021, [Accessed 2 January 2021].

[134] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h. 264/avc video coding standard,” *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[135] Y. Chen, D. Mukherjee, J. Han, A. Grange, Y. Xu, S. Parker, C. Chen, H. Su, U. Joshi, C.-H. Chiang *et al.*, “An overview of coding tools in av1: the first video codec from the alliance for open media,” *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[136] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi *et al.*, “An overview of core coding tools in the av1 video codec,” in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 41–45.

[137] LambdaTest, “AVIF Image Format - The Next-Gen Compression Codec,” https://www.lambdatest.com/blog/avif-image-format/, 2021, [Accessed 2 January 2021].

[138] En.wikipedia.org, “AV1,” https://en.wikipedia.org/wiki/AV1, 2021, [Accessed 19 January 2021].

[139] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[140] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.

[141] D. Salomon and G. Motta, *Handbook of data compression*. Springer Science & Business Media, 2010.

[142] J. Ding, J. C. Furgeson, and E.-M. Sha, “Application specific image compression for virtual conferencing,” in *Proceedings International Conference on Information Technology: Coding and Computing (Cat. No. PR00540)*. IEEE, 2000, pp. 48–53.

[143] S. Bhavani and K. Thanushkodi, “A survey on coding algorithms in medical image compression,” *International Journal on Computer Science and Engineering*, vol. 2, no. 5, pp. 1429–1434, 2010.

[144] G. K. Kharate and V. H. Patil, “Color image compression based on wavelet packet best tree,” *arXiv preprint arXiv:1004.3276*, 2010.

[145] M. Haque and F. Ahmed, “Image data compression with jpeg and jpeg2000,” *8th International Confrrence on Computer and Information Technology*, pp. 1064–1069, 2005.

[146] M. A. Joshi, *Digital image processing: An algorithmic approach*. PHI Learning Pvt. Ltd., 2018.

[147] S. Golomb, “Run-length encodings (corresp.),” *IEEE transactions on information theory*, vol. 12, no. 3, pp. 399–401, 1966.

[148] W. Burger and M. J. Burge, *Digital image processing: an algorithmic introduction using Java*. Springer, 2016.

[149] S. Benndorf, “Method for the compression of data using a run-length coding,” Feb. 12 2013, uS Patent 8,374,445.

[150] S. Shanmugasundaram and R. Lourdusamy, “A comparative study of text compression algorithms,” *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68–76, 2011.

[151] S. Kodituwakku and U. Amarasinghe, "Comparison of lossless data compression algorithms for text data," *Indian journal of computer science and engineering*, vol. 1, no. 4, pp. 416–425, 2010.

[152] M. A. Rahman, S. M. S. Islam, J. Shin, and M. R. Islam, "Histogram alternation based digital image compression using base-2 coding," in *2018 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2018, pp. 1–8.

[153] J. H. Pujar and L. M. Kadlaskar, "A new lossless method of image compression and decompression using huffman coding techniques." *Journal of Theoretical & Applied Information Technology*, vol. 15, 2010.

[154] M. K. Mathur, S. Loonker, and D. Saxena, "Lossless huffman coding technique for image compression and reconstruction using binary trees," *International Journal of Computer Technology and Applications*, vol. 3, no. 1, 2012.

[155] G. Vijayvargiya, S. Silakari, and R. Pandey, "A survey: various techniques of image compression," *arXiv preprint arXiv:1311.6877*, 2013.

[156] M. A. Rahman, J. Shin, A. K. Saha, and M. R. Islam, "A novel lossless coding technique for image compression," in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2018, pp. 82–86.

[157] A. M. Rufai, G. Anbarjafari, and H. Demirel, "Lossy medical image compression using huffman coding and singular value decomposition," in *2013 21st Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2013, pp. 1–4.

[158] R. P. Jasmi, B. Perumal, and M. P. Rajasekaran, "Comparison of image compression techniques using huffman coding, dwt and fractal algorithm," in *2015 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2015, pp. 1–5.

[159] A. Masmoudi and A. Masmoudi, "A new arithmetic coding model for a block-based lossless image compression based on exploiting inter-block correlation," *Signal, Image and Video Processing*, vol. 9, no. 5, pp. 1021–1027, 2015.

[160] X. Li and M. T. Orchard, "Edge-directed prediction for lossless compression of natural images," *IEEE Transactions on image processing*, vol. 10, no. 6, pp. 813–817, 2001.

[161] H. Yokoo, "Improved variations relating the ziv-lempel and welch-type algorithms for sequential data compression," *IEEE transactions on information theory*, vol. 38, no. 1, pp. 73–81, 1992.

[162] C. Saravanan and M. Surender, "Enhancing efficiency of huffman coding using lempel ziv coding for image compression," *International journal of soft computing and Engineering*, vol. 2, no. 6, pp. 38–41, 2013.

[163] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[164] Osirix-viewer.com, "DICOM Image Library," https://www.osirix-viewer.com/resources/dicom-image-library/, 2019, [Accessed 19 January 2020].

[165] I. Schiopu and A. Munteanu, "Deep-learning-based lossless image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 1829–1842, 2019.

[166] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[167] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE transactions on Communications*, vol. 45, no. 4, pp. 437–444, 1997.

[168] I. Schiopu and A. Munteanu, "Macro-pixel prediction based on convolutional neural networks for lossless compression of light field images," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 445–449.

[169] ——, "Residual-error prediction based on deep learning for lossless image compression," *Electronics Letters*, vol. 54, no. 17, pp. 1032–1034, 2018.

[170] H. Rhee, Y. I. Jang, S. Kim, and N. I. Cho, "Lossless image compression by joint prediction of pixel and context using duplex neural networks," *IEEE Access*, 2021.

[171] F. Bellard, "Bpg image format," *URL https://bellard. org/bpg*, vol. 1, p. 2, 2015.

[172] T. Boutell and T. Lane, "Png (portable network graphics) specification version 1.0," *Network Working Group*, pp. 1–102, 1997.

[173] S. Kim and N. I. Cho, "Hierarchical prediction and context adaptive coding for lossless color image compression," *IEEE Transactions on image processing*, vol. 23, no. 1, pp. 445–449, 2013.

[174] W. B. P. Version, "1.0 hardware reference guide, xp-002202892, network engines," *Inc., Jun*, vol. 1, p. 92, 2000.

[175] J. Alakuijala, R. van Asseldonk, S. Boukortt, M. Bruse, I.-M. Comșa, M. Firsching, T. Fischbacher, E. Kliuchnikov, S. Gomez, R. Obryk *et al.*, "Jpeg xl next-generation image compression architecture and coding tools," in *Applications of Digital Image Processing XLII*, vol. 11137. International Society for Optics and Photonics, 2019, p. 111370K.

[176] H. Rhee, Y. I. Jang, S. Kim, and N. I. Cho, "Channel-wise progressive learning for lossless image compression," in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 1113–1117.

[177] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," *arXiv preprint arXiv:1606.05328*, 2016.

[178] S. Reed, A. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. Freitas, "Parallel multiscale autoregressive density estimation," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2912–2921.

[179] E. Hoogeboom, J. W. Peters, R. v. d. Berg, and M. Welling, "Integer discrete flows and lossless compression," *arXiv preprint arXiv:1905.07376*, 2019.

[180] H. Ma, D. Liu, N. Yan, H. Li, and F. Wu, "End-to-end optimized versatile image compression with wavelet-like transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[181] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[182] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” in *International Conference on Machine Learning*. PMLR, 2015, pp. 2256–2265.

[183] L. Dinh, D. Krueger, and Y. Bengio, “Nice: Non-linear independent components estimation,” *arXiv preprint arXiv:1410.8516*, 2014.

[184] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, “Draw: A recurrent neural network for image generation,” in *International Conference on Machine Learning*. PMLR, 2015, pp. 1462–1471.

[185] A. van den Oord and J. Dambre, “Locally-connected transformations for deep gmms,” in *International Conference on Machine Learning (ICML): Deep learning Workshop*, 2015, pp. 1–8.

[186] K. Gregor, F. Besse, D. Jimenez Rezende, I. Danihelka, and D. Wierstra, “Towards conceptual compression,” *Advances In Neural Information Processing Systems*, vol. 29, pp. 3549–3557, 2016.

[187] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using real nvp,” *arXiv preprint arXiv:1605.08803*, 2016.

[188] N. Kalchbrenner, A. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, “Video pixel networks,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 1771–1779.

[189] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, “Improved variational inference with inverse autoregressive flow,” *Advances in neural information processing systems*, vol. 29, pp. 4743–4751, 2016.

[190] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *International Conference on Machine Learning*. PMLR, 2016, pp. 1747–1756.

[191] I. Schiopu, H. Huang, and A. Munteanu, “Cnn-based intra-prediction for lossless hevc,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 1816–1828, 2019.

[192] F. Mentzer, L. V. Gool, and M. Tschannen, “Learning better lossless compression using lossy compression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6638–6647.

[193] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit *et al.*, “Openimages: A public dataset for large-scale multi-label and multi-class image classification,” *Dataset available from https://github.com/openimages*, vol. 2, no. 3, p. 18, 2017.

[194] E. Agustsson and R. Timofte, “Ntire 2017 challenge on single image super-resolution: Dataset and study,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 126–135.

[195] J. Li, J. Wu, and G. Jeon, “Gpu acceleration of clustered dpcm for lossless compression of hyperspectral images,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 2906–2916, 2019.

[196] M. Li, K. Ma, J. You, D. Zhang, and W. Zuo, “Efficient and effective context-based convolutional entropy modeling for image compression,” *IEEE Transactions on Image Processing*, vol. 29, pp. 5900–5911, 2020.

[197] T. Suzuki, “Wavelet-based spectral–spatial transforms for cfa-sampled raw camera image compression,” *IEEE Transactions on Image Processing*, vol. 29, pp. 433–444, 2019.

[198] P. Chrabaszcz, I. Loshchilov, and F. Hutter, “A downsampled variant of imagenet as an alternative to the cifar datasets,” *arXiv preprint arXiv:1707.08819*, 2017.

[199] Oomo.com, “Becoming a data-driven CEO | Domo,” https://www.domo.com/solution/data-never-sleeps-6, 2020, [Accessed 19 January 2020].

[200] W. Pan, Z. Li, Y. Zhang, and C. Weng, “The new hardware development trend and the challenges in data management and analysis,” *Data Science and Engineering*, vol. 3, no. 3, pp. 263–276, 2018.

[201] I. Blanes, E. Magli, and J. Serra-Sagrista, “A tutorial on image compression for optical space imaging systems,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 2, no. 3, pp. 8–26, 2014.

[202] F. Liu, M. Hernandez-Cabronero, V. Sanchez, M. W. Marcellin, and A. Bilgin, “The current role of image compression standards in medical imaging,” *Information*, vol. 8, no. 4, p. 131, 2017.

[203] E. Syahrul, “Lossless and nearly-lossless image compression based on combinatorial transforms,” Ph.D. dissertation, Université de Bourgogne, 2011.

[204] Y. Deigant, V. Akshat, H. Raunak, P. Pranjal, and J. Avi, “A proposed method for lossless image compression in nano-satellite systems,” in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–11.

[205] S.-G. Miaou, F.-S. Ke, and S.-C. Chen, “A lossless compression method for medical image sequences using jpeg-ls and interframe coding,” *IEEE transactions on information technology in biomedicine*, vol. 13, no. 5, pp. 818–821, 2009.

[206] J. Taquet and C. Labit, “Hierarchical oriented predictions for resolution scalable lossless and near-lossless compression of ct and mri biomedical images,” *IEEE Transactions on image processing*, vol. 21, no. 5, pp. 2641–2652, 2012.

[207] S. S. Parikh, D. Ruiz, H. Kalva, G. Fernández-Escribano, and V. Adzic, “High bit-depth medical image compression with hevc,” *IEEE journal of biomedical and health informatics*, vol. 22, no. 2, pp. 552–560, 2017.

[208] J. Lee, J. Yun, J. Lee, I. Hwang, D. Hong, Y. Kim, C. G. Kim, and W.-C. Park, “An effective algorithm and architecture for the high-throughput lossless compression of high-resolution images,” *IEEE Access*, vol. 7, pp. 138 803–138 815, 2019.

[209] J. ITU-T, “Xr image coding system–image coding specification,” *ITU-T Recommendation*, vol. 832, 2009.

[210] J. Shukla, M. Alwani, and A. K. Tiwari, “A survey on lossless image compression methods,” in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 6. IEEE, 2010, pp. V6–136.

[211] C. Zhang and T. Chen, “A survey on image-based rendering—representation, sampling and compression,” *Signal Processing: Image Communication*, vol. 19, no. 1, pp. 1–28, 2004.

[212] O. U. G. Programs, “How Much Data Is Produced Every Day?” https://www.northeastern.edu/graduate/blog/how-much-data-produced-every-day/, 2020, [Accessed on 17 September 2020].

[213] B. Walker, “Every day big data statistics—2.5 quintillion bytes of data created daily.VCloudNews ,” http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily, 2015, [Accessed on 17 September 2020].

[214] Blog.microfocus.com, “How Much Data Is Created on The Internet Each Day? Micro Focus Blog,” https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/, 2020, [Accessed on 18 May 2020].

[215] H. Larkin, “Word indexing for mobile device data representations,” in *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*. IEEE, 2007, pp. 399–404.

[216] I. M. Pu, *Fundamental data compression*. Butterworth-Heinemann, 2005.

[217] S. Porwal, Y. Chaudhary, J. Joshi, M. Jain *et al.*, “Data compression methodologies for lossless data and comparison between algorithms,” *International Journal of Engineering Science and Innovative Technology (IJESIT) Volume*, vol. 2, pp. 142–147, 2013.

[218] M. Saračević, S. Adamović, and E. Biševac, “Application of catalan numbers and the lattice path combinatorial problem in cryptography,” *Acta Polytechnica Hungarica*, vol. 15, no. 7, pp. 91–110, 2018.

[219] M. Saračević, S. Adamović, V. Miškovic, N. Maček, and M. Šarac, “A novel approach to steganography based on the properties of catalan numbers and dyck words,” *Future Generation Computer Systems*, vol. 100, pp. 186–197, 2019.

[220] M. Pandey, S. Shrivastava, S. Pandey, and S. Shridevi, “An enhanced data compression algorithm,” in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, 2020, pp. 1–4.

[221] C. Oswald and B. Sivaselvan, “An optimal text compression algorithm based on frequent pattern mining,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 3, pp. 803–822, 2018.

[222] J. Portell, R. Iudica, E. García-Berro, A. Villafranca, and G. Artigues, “Fapec, a versatile and efficient data compressor for space missions,” *International journal of remote sensing*, vol. 39, no. 7, pp. 2022–2042, 2018.

[223] R. Rahim, “Combination of the blowfish and lempel-ziv-welch algorithms for text compression,” 2017.

[224] A. Gupta, A. Bansal, and V. Khanduja, “Modern lossless compression techniques: Review, comparison and analysis,” in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2017, pp. 1–8.

[225] T. A. Welch, “Technique for high-performance data compression,” *Computer*, no. 52, 1984.

[226] D. Salomon, *A concise introduction to data compression*. Springer Science & Business Media, 2007.

[227] M. Nelson and J. Gailly, *The data compression book 2nd edition*. M T Books, New York, NY, 1995.

[228] En.wikipedia.org, "HLZ77 And LZ78," https://en.wikipedia.org/wiki/LZ77_and_LZ78, 2020, [Accessed on 27 May 2020].

[229] ——, "Burrows–Wheeler Transform," https://en.wikipedia.org/wiki/Burrows_Wheeler_transform, 2020, [Accessed on 27 May 2020].

[230] I. M. El-Henawy, E. R. Mohamed, N. A. Lashin *et al.*, "A hybrid technique for data compression," *International Journal of Digital Content Technology and its Applications*, vol. 9, no. 2, p. 11, 2015.

[231] H. Kaur and B. Jindal, "Lossless text data compression using modified huffman coding-a review," in *Proceedings of the International Conference on Technologies for Sustainability-Engineering, Information Technology, Management and the Environment*. Citeseer, 2015, pp. 1017–1025.

[232] V. T. Todorov, R. K. Kountchev, M. G. Milanova, R. A. Kountcheva, and C. W. Ford Jr, "Method and apparatus for lossless run-length data encoding," Apr. 29 2008, uS Patent 7,365,658.

[233] P. G. Howard and J. S. Vitter, "New methods for lossless image compression using arithmetic coding," *Information processing & management*, vol. 28, no. 6, pp. 765–779, 1992.

[234] F. S. Awan and A. Mukherjee, "Lipt: A lossless text transform to improve compression," in *Proceedings International Conference on Information Technology: Coding and Computing*. IEEE, 2001, pp. 452–460.

[235] G. Manzini, "The burrows-wheeler transform: theory and practice," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1999, pp. 34–47.

[236] D. Adjeroh, T. Bell, and A. Mukherjee, *The Burrows-Wheeler Transform:: Data Compression, Suffix Arrays, and Pattern Matching*. Springer Science & Business Media, 2008.

[237] 7-zip.org, "7Z Format," https://www.7-zip.org/7z.html, 2020, [Accessed on 25 August 2020].

[238] R. A. Patel, Y. Zhang, J. Mak, A. Davidson, and J. D. Owens, *Parallel lossless data compression on the GPU*. IEEE, 2012.

[239] M. Mahoney, "Large Text Compression Benchmark," http://mattmahoney.net/dc/text.html, 2020, [Accessed on 7 September 2020].

[240] ——, "Data Compression Programs," http://www.mattmahoney.net/dc/, 2020, [Accessed on 7 September 2020].

[241] J. Alakuijala and Z. Szabadka, "Brotli compressed data format," *Internet Engineering Task Force*, 2016.

[242] Theregister.com, "Google's New Squeeze: Brotli Compression Open-Sourced," https://www.theregister.com/2015/09/23/googles_brotli_compression_opensourced, 2020, [Accessed on 7 August 2020].

[243] J. Alakuijala, E. Kliuchnikov, Z. Szabadka, and L. Vandevenne, "Comparison of brotli, deflate, zopfli, lzma, lzham and bzip2 compression algorithms," *Google Inc*, 2015.

[244] D. Salomon, "Data compression," in *Handbook of massive data sets*. Springer, 2002, pp. 245–309.

[245] Corpus.canterbury.ac.nz, "The Canterbury Corpus," http://corpus.canterbury.ac.nz/, 2020, [Accessed on 30 May 2020].

[246] P. Gage, "A new algorithm for data compression," *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.